Ville Ahonen

Tiedonsiirto sisällönhallintajärjestelmistä pilviympäristöön

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
18.10.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Ville Ahonen Tiedonsiirto sisällönhallintajärjestelmistä pilviympäristöön 54 sivua + 6 liitettä 18.10.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	johtaja Matias Ärje yliopettaja Kari Aaltonen

Insinöörityössä suunniteltiin ja toteutettiin ohjelmistoyritykselle tiedonsiirtomoduuli, joka toimii osana pilvijulkaisualustaa. Pilvijulkaisualusta esittää eri sisällönhallintajärjestelmien sinne lähettämää sisältöä. Tämä sisältö siirretään julkaisualustalle tiedonsiirtomoduulin avulla. Pilvijulkaisualusta on osa ohjelmistoyrityksen strategista suuntausta, jossa sovelluksia aletaan kehittämään pilviympäristöön. Pilvijulkaisualustan toteutusympäristönä toimi pilvipalvelualusta Windows Azure Platform. Moduuli toteutettiin WWW-sovelluspalveluna hyödyntäen Windows Communication Foundation alustaa, jolla voidaan luoda WWW-sovelluspalveluita Microsoftin .NET-ympäristössä.

Työssä myös tutkittiin erilaisia pilvijärjestelmiä painottuen toteutusalustaksi valittuun Windows Azure Platformiin. Alustan käyttöä ja sen tuomia haasteita tutkittiin sovelluskehityksen näkökulmasta. Lisäksi pyrittiin löytämään kohtia, joihin sovelluskehityksessä tulisi erityisesti kiinnittää huomiota uudentyyppistä alustaa käytettäessä.

Tutkimuksen tuloksena huomattiin, että ominaisuuksiltaan erilaisia pilvijärjestelmiä on useita. Pilvijärjestelmien keskinäiset erot painottuvat lähinnä siihen, kuinka paljon niiden ominaisuuksiin voidaan vaikuttaa. Lisäksi eroja on järjestelmien tukemissa ohjelmointikielissä ja teknologioissa. Windows Azure Platformin tarkastelussa nostettiin esiin muutamia huomionarvoisia kohtia. Tärkeimpänä on se, että heti suunnittelusta lähtien tulee kiinnittää huomiota pilvessä ajettavan sovelluksen aiheuttamiin kustannuksiin. Alustan laskutus on käyttömääräperusteinen ja tämä johtaa siihen, että melko pienetkin tekniset ratkaisut saattavat vaikuttaa merkittävästi sovelluksen kustannuksiin. Työssä tuli esille erityisesti, että taulutallennusratkaisua käytettäessä syntyy helposti turhia kustannuksia, joilta voidaan välttyä optimoimalla tallennusratkaisuiden käyttöä.

Avainsanat	Azure, pilvilaskenta, tiedonsiirto, WWW-sovelluspalvelu

Metropolia

Abstract

Author Title Number of Pages Date	Data transfer from content distribution service to cloud environment 54 pages + 6 appendices 18 October 2011
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Matias Ärje, Director Kari Aaltonen, Principal Lecturer

This engineering thesis consisted of designing and implementing a data transfer module for a software company. The data transfer module is a part of a cloud based publishing platform. The publishing platform is used to present content sent by different content management systems. This cloud based publishing platform is a part of a new strategy that the software company is adapting. The strategy consists of developing software for cloud environment. The publishing platform was built upon Windows Azure Platform that is a cloud platform. The data transfer module was built as a web service and the implementation used Windows Communication Foundation platform that is web service platform for Microsofts .Net environment.

Different cloud systems where also studied. A special attention was given to the Windows Azure Platform it being the platform used for developing the publishing platform. The usage of the platform and the challenges it creates were being studied from the perspective of software development. One goal was also to recognize parts that need some special attention when doing software development for this new kind of platform.

The results of the study show that there are numerous cloud systems that vary by their properties. The biggest differences between the systems are in how much you can customize their properties. Also there are differences between the supported programming languages and technologies. Few important subjects were highlighted when studying Windows Azure Platform. The most important thing is that the cost of running a software in the cloud must be taken in to account right away in the design phase. The pricing of the platform is usage based and as a result of this even minute technical solutions can have a major impact on the running costs of the application. In this thesis it was especially noticed that it is easy to create unnecessary costs when using the table storage and that this can be avoided by optimizing the way table storage is used.

Keywords	Azure, cloud computing, data transfer, web service

Sisällys

Lyhenteet, käsitteet ja määritelmät

HTTP/HTTPS	Hyper Text Transfer Protocol tai Hyper Text Transfer Protocol Secure. Protokolla, jonka avulla WWW-palvelimet ja asiakkaat siirtävät tietoa.
Cloud computing	Pilvilaskenta on tilanne, jossa erilaisia tietotekniikan resursseja käytetään verkkoyhteyden avulla.
CMS	Content Management System eli sisällönhallintajärjestelmä on tietojärjestelmä, jolla hallitaan erilaisia sisältöjä.
Moduuli	Ohjelmistokehityksessä moduulilla tarkoitetaan ohjelman itsenäistä osaa.
REST	Representational State Transfer on http-protokollaa käyttävä arkkitehtuurimalli, jossa palvelin tarjoaa asiakkaille resursseja käytettäväksi erilaisten http-pyyntöjen perusteella. Resursseilla voi olla erilaisia esitysmuotoja. Asiakkaan pyynnöt voivat muokata resurssin tilaa, ja vastaavasti asiakkaat voivat päivittää oman tilansa resurssin perustella.
SPI-malli	Software Platform Infrastructure -mallissa pilvijärjestelmän luokitteluun käytetään kolmea eri tasoa. Tasot ovat sovellus, alusta ja infrastruktuuri.
URL-osoite	Universal Resource Locator on merkkijono, jonka avulla internetissä oleva resurssi voidaan löytää ja noutaa.
Web Service	WWW-sovelluspalvelu on tietoverkossa sijaitseva ohjelmistojärjestelmä, jonka palveluita muut verkossa sijaitsevat yhteensopivat järjestelmät voivat hyödyntää.

1 Johdanto

Insinöörityön tavoitteena on suunnitella ja toteuttaa ohjelmistoyritykselle pilviteknologiaa käyttävä tiedonsiirtomoduuli. Moduuli toimii osana yrityksen uutta pilvipohjaista julkaisualustaa. Julkaisualusta tulee esittämään internetissä eri sisällönhallintajärjestelmien tarjoamaa sisältöä WWW-sivujen muodossa. Moduulia käytetään näiden sisältöjen siirtämiseen julkaisualustalle. Tätä varten moduuli tarjoaa internetissä julkisen rajapinnan, johon sisällönhallintajärjestelmät voivat lähettää sisältöä. Julkaisualusta toteutetaan Windows Azure Platform pilvipalvelualustan avulla. Windows Azure Platform on Microsoftin vuonna 2009 julkaisema pilvipalvelualusta, jonka käyttöön ohjelmistoyritys on Microsoftin kumppanina sitoutunut.

Ohjelmistoyritys toimittaa erilaisia WWW-pohjaisia ohjelmistoja yrityksille ja yhteisöille. Pilvipalveluna tarjottava julkaisualusta on ensimmäinen osa ohjelmistoyrityksen uutta strategiaa, jossa sovelluksia pyritään siirtämään pilvialustalle. Sovellusten tarjoaminen pilvipalveluina helpottaa liiketoiminnan laajentamista paikallisilta markkinoilta globaaleille. Strategiaan kuuluu myös ohjelmistojen muodostaminen julkaisualustan kaltaisista pienehköistä pilvikomponenteista, joita voidaan tarpeen mukaan yhdistellä laajempien kokonaisuuksien toteuttamiseksi. Julkaisualustan ensimmäinen versio on tarkoitus esitellä Los Angelesissa pidettävässä Microsoftin kumppanuustapahtumassa heinäkuussa 2011.

Insinöörityössä tutustutaan pilvilaskentaan ja erilaisiin pilvijärjestelmiin. Järjestelmistä tarkemmin perehdytään Windows Azure Platformiin ja sen käyttöä tutkitaan sovelluskehityksen näkökulmasta. Tarkastelussa ovat alustan mahdollisesti aiheuttamat haasteet jotka tulee huomioida sovelluksia kehitettäessä. Tarkoituksena on myös selvittää, kuinka helposti uudentyyppinen alusta on sovelluskehittäjien omaksuttavissa ja vaatiiko se paljon opiskelua. Lisäksi pyritään löytämään hyviä käytänteitä, joita sovelluskehitys pilvipalvelualustalle vaatii.

2 Pilvilaskenta

Termi pilvilaskenta (cloud computing) on melko lavea eikä sille vielä ole muodostunut kovinkaan täsmällistä määritelmää. Useimmissa lähteissä pilvilaskennalla tarkoitetaan järjestelyä, jossa erinäisiä tietotekniikan resursseja käytetään verkkoyhteyden avulla. Termillä pilvi kuvataan sitä, että resurssit ovat helposti saavutettavissa, mutta niiden fyysistä sijaintia ja toimintatapaa ei tiedetä. Tässä tutkielmassa pilvilaskennalla tarkoitetaan sitä kokonaisuutta, jossa pilvestä saatavaa resurssia käytetään. Pilvilaskennan tärkeimmiksi tekijöiksi voidaan nimetä resurssien virtuaalisuus, käyttömääräperusteinen hinnoittelu ja skaalautuvuus [1, s. 50–51].

Virtuaalisilla resursseilla tarkoitetaan sitä, että resursseja voidaan käyttää yksinkertaisesti ja välittämättä siitä kuinka niitä tuotetaan. Tilannetta voi verrata vaikkapa veden käyttöön ja vesijohtoverkkoon. Kun tarvitaan vettä, etsitään lähin vesihana ja valutetaan sieltä vettä. Ideaalitilanteessa vesi on riittävän puhdasta eikä vedenkäyttäjän tarvitse pohtia, tuliko vesi järvestä, kaivosta vai merestä ja pumpattiinko se hanaan vesipumpulla vai vesitornista. [1, s. 3.] Tyypillisiä pilvestä käytettäviä resursseja ovat tallennus- ja laskentakapasiteetti sekä näitä hyödyntävät kokonaiset sovellukset, kuten verkkopohjainen Google Docs -toimisto-ohjelmisto.

Käyttömääräperusteinen hinnoittelu tarkoittaa sitä, että maksetaan ainoastaan siitä mitä käytetään. Tällainen hinnoittelu mahdollistaa pilvilaskennan käytön lähes olemattomilla aloituskustannuksilla. Tämä ominaisuus tekee pilvilaskennan hyödyntämisen erittäin houkuttelevaksi esimerkiksi startup-yrityksille, joilla ei ole suurta alkupääomaa.

Resurssien virtuaalisuus ja käyttömääräperusteinen hinnoittelu johtavat yhdessä skaalautuvuuteen. Käytännössä resursseja voidaan aina hankkia tarpeen mukaan lisää silloin kuin tarvetta ilmenee. Vastaavasti käyttämättömät resurssit voidaan vapauttaa ja näin säästyä turhilta kuluilta. Näin myös pilvilaskentaa tarjoavat yritykset voivat hyödyntää laitteistoaan mahdollisimman tehokkaasti.

2.1 Historia

Yksi tapa tutkia pilvilaskennan historiaa on katsoa niitä tietojenkäsittelyjärjestelmiä, jotka ovat vaikuttaneet pilvilaskentaan. Nykyisellään pilvilaskenta koostuu useiden erilaisten tietojenkäsittelyjärjestelmien yhdistelmästä. Näitä järjestelmiä ovat muun muassa aikajako, keskustietokone, transaktio ja grid-järjestelmät. [2, s. 2.] Yhdessä nämä järjestelmät esittelivät ja mahdollistivat pilvilaskennalle tyypillisiä ominaisuuksia, kuten hyvän skaalautuvuuden ja käyttömääräperusteisen hinnoittelun.

Aikajakojärjestelmät yleistyivät 1960-luvulla. Ennen tätä ohjelmat suoritettiin reikäkorttien ja nauhojen avulla yhdellä tietokoneella peräjälkeen. Tämä oli tehotonta, koska tietokone saattoi olla ohjelmien suoritusten välillä käyttämättä eikä sen koko kapasiteettia saatu hyödynnettyä. Aikajakojärjestelmien perusideana oli, että tietokoneen laskenta-aikaa jaetaan sitä tarvitseville ohjelmille paloissa. Näin esimerkiksi yhden ohjelman odottaessa käyttäjän syötettä, voidaan tietokonetta käyttää samalla jonkun muun ohjelman suorittamiseen. Tätä laskenta-aikaa alettiin myymään sitä tarvitseville, ja jotkut yritykset jopa tarjosivat laskenta-aikaa puhelinverkon välityksellä. Keskustietokoneet toimivat samalla periaatteella kuin aikajakojärjestelmät. Tehokkaan ja kalliin tietokoneen resursseja jaettiin useiden käyttäjien kesken. Tämä laskenta-ajan ostaminen tarpeen mukaan on yksi nykyisten pilvijärjestelmien perusominaisuuksista. Keskustietokoneet kehittyivät aikajakojärjestelmistä eteenpäin hyödyntämällä muun muassa virtualisointia. Virtualisoinnin avulla yhdelle tietokoneelle voitiin asentaa useita virtuaalisia käyttöjärjestelmiä, jotka toimivat yhden pääkäyttöjärjestelmän sisällä. Virtualisointi toimii monissa pilvijärjestelmissä keskeisessä roolissa. [2, s. 2–3.] Sen avulla voidaan jakaa laitteistoresursseja tarvittavan kokoisissa osissa käyttäjille ja näin hyödyntää mahdollisimman suuri osa laitteiston kapasiteetista.

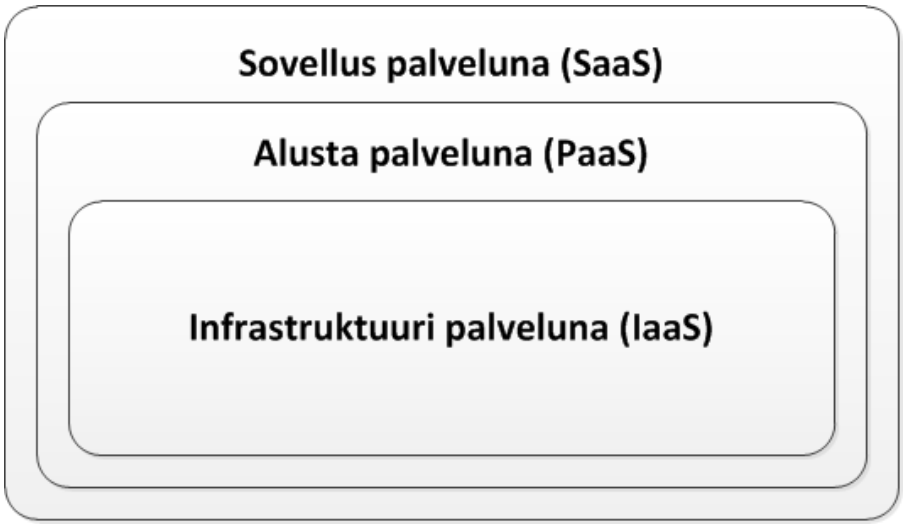
Transaktiojärjestelmät ovat järjestelmiä, joissa suoritetut tehtävät tai tehtäväkokonaisuudet ovat atomisia. Tämä tarkoittaa sitä, että ne joko onnistuvat kokonaan tai epäonnistuvat kokonaan eikä keskeneräinen tila näy järjestelmän muille käyttäjille. Atomisuuden avulla voidaan suorittaa monimutkaisia tehtäväketjuja ilman pelkoa siitä, että järjestelmä ajautuu epävakaaseen tilaan jonkun osatehtävän epäonnistuessa. [2, s. 4–5.] Esimerkkinä tällaisesta tilanteesta on tilisiirto pankkijärjestelmässä. Kun rahaa siirretään, varmistetaan ensin, että tilillä on rahaa. Tämän jälkeen lisätään siirrettävä summa halutulle tilille. Lopuksi alkuperäiseltä tililtä vähennetään siirretty summa. Jos

mikä tahansa vaihe tilisiirrossa epäonnistuu, palautuu tilanne alkuperäiseksi. Atomisuus siis mahdollistaa järjestelmän luotettavan toiminnan tilanteessa, jossa järjestelmällä on useita yhtäaikaisia käyttäjiä. Transaktiojärjestelmien kaltainen luotettavuus on erittäin tärkeää hajautetuissa pilvijärjestelmissä, joissa useat käyttäjät käyttävät samoja resursseja samaan aikaan [2, s. 5]. Vahvasti hajautetuissa järjestelmissä joudutaan kuitenkin aina tekemään kompromisseja järjestelmän oikeellisen toiminnan ja palvelun saavutettavuuden välillä [3].

Grid-laskennalla tarkoitetaan suurien tietojenkäsittelytehtävien pilkkomista osiin, jolloin osia voidaan käsitellä usealla tietokoneella yhtä aikaa. Lopuksi käsitellyt osat voidaan yhdistää, ja näin tehtävä saadaan ratkaistuksi nopeammin. Usean heikompitehoisen tietokoneen käyttäminen yhden tehokkaan sijaan on monesti myös kustannustehokkaampaa. Termillä grid viitataankin verkkoon, jonka muodostavat pilkottuja tehtäviä käsittelevät tietokoneet. Verkon muodostamiseen voidaan käyttää esimerkiksi internetiä. Klassinen esimerkki grid-laskennasta on SETI@home -projekti, jonka avulla kuka tahansa voi osallistua maan ulkopuolisen elämän etsimiseen hyödyntämällä tietokoneensa laskentakapasiteettia teleskooppien keräämän datan analysoimiseen. Tällainen usean tietokoneen työskentely yhteisen tehtävän hyväksi on yksi pilvilaskennan kulmakivistä. Lähes kaikki pilvijärjestelmät mahdollistavat kapasiteetin kasvattamisen monistamalla laskentaa suorittavia osia. [2, s. 4–6.]

2.2 Pilvijärjestelmien luokittelu

Erilaisten pilvipohjaisten järjestelmien luokitteluun käytetään usein niin sanottua SPI-mallia, jossa kerrokset kuvaavat, millä abstraktiotasolla pilvilaskentaa käytetään. Kerrosten voidaan myös katsoa rakentuvan toistensa päälle siten, että ylempi kerros käyttää alempia hyödykseen. [1, s. 50–21; 4, s. 13–16.] Kuviossa 1 esitetään SPI-malli.



Kuvio 1. SPI-malli

Kuviosta selviää, kuinka SPI-mallin tasot muodostavat kerrosrakenteen, jossa uloimmat kerrokset rakentuvat alempien ympärille. Alimpana on infrastruktuurin tarjoaminen palveluna. Tämän päälle rakentuu alustan tarjoaminen palveluna. Sovelluksen tarjoaminen palveluna muodostaa mallin uloimman kerroksen.

Infrastruktuurin tarjoaminen palveluna (infrastructure as a service) eli IaaS on abstraktiotasoista matalimmalla. IaaS tarkoittaa sitä, että pilvestä on hankittavissa virtuaalisia tietotekniikan ”raakaresursseja”, kuten mahdollisuus datan tallentamiseen, tiedonsiirtoon tai laskenta-aikaan. Käytännössä useimmat IaaS-tasoa tarjoavat palveluntarjoajat myyvät verkon välityksellä käytettäviä virtuaalikoneita, joihin käyttäjä voi asentaa tai ohjelmoida haluamansa toiminnot. [1, s. 51; 4, s. 13–14.]

Sovellusalusta palveluna (platform as a service) eli PaaS nostaa abstraktiotasoa siten, että infrastruktuuriresurssien käyttö ja allokointi on tehty erittäin helpoksi ja jopa automaattiseksi. Ohjelmistokehityksen näkökulmasta PaaS tarjoaa yksinkertaiset rajapinnat, joiden avulla voidaan ohjelmoida sovelluksia nimenomaan pilviympäristöön. [4, s. 14; 5, s. 7–8.]

Sovellus palveluna (software as a service) eli SaaS kuvaa kokonaisten sovellusten tarjoamista palvelun muodossa [1, s. 51]. Vaihtoehtoinen termi SaaS-sovellukselle on pilvisovellus. Loppukäyttäjä käyttää useimmiten pilvisovellusta WWW-selaimen välityksellä [5, s. 5–6]. Etuna tässä mallissa loppukäyttäjälle on se, että sovelluksia päivite-

tään useimmiten automaattisesti. Näin sovellusten ylläpitämiseen ei tarvitse käyttää omia resursseja. Toinen tärkeä säästökohde muodostuu siitä, että teknisesti raskaitakin sovelluksia voidaan käyttää heikkotehoisemmalla päätelaitteella, koska itse laskenta suoritetaan pilvessä eikä päätelaitteella kuten perinteisesti [5, s. 5–6]. Lisäksi sovelluksiin voidaan hankkia käyttöoikeuksia erittäin joustavasti ja nopeasti. Esimerkkejä pilvisovelluksista ovat muun muassa toimisto-ohjelmisto Google Docs, tiedontallennuspalvelu DropBox ja sähköposti Google Gmail.

2.3 Pilvipalvelualustat

Pilvipalvelualusta on PaaS-tasolla toimiva järjestelmä, jonka avulla palvelutarjoaja mahdollistaa sovellusten kehittämisen heidän pilvialustalleen. Ohjelmistokehittäjien näkökulmasta pilvipalvelualusta on se ympäristö, johon he ohjelmansa suunnittelevat ja toteuttavat ja jonka eri palveluita käyttämällä saadaan pilvilaskennan hyödyt esille. Pilvipalvelualustoja on useita, ja niiden määrä näyttää kasvavan alati. Tällä hetkellä ehkä merkittävimmät ja lupaavimmat alustat ovat Amazon Web Services, Google App Engine ja Windows Azure Platform.

Amazon Web Services (AWS)

Amazon Web Services sisältää suuren joukon Amazon–osakeyhtiön tarjoamia pilvikomponentteja. Sähköiseen kaupankäyntiin erikoistunut Amazon aloitti toimintansa verkkokauppana vuonna 1995 [6]. Amazonin verkkokauppaliiketoiminta lähti nopeaan kasvuun, ja vuonna 2002 Amazon alkoi kehitellä pilvipohjaisia infrastruktuuriratkaisuita omien tarpeittensa pohjalta.

Amazon on pilvipalvelualustojen edelläkävijä ja nykyisen alustan keskeisimmät osat ovat:

- viestien välitykseen tarkoitettu jonopalvelu Amazon Simple Queue Service (SQS) [7, s. 1–8]
- tiedostotallennuskapasiteettia tarjoava Amazon Simple Storage Service (S3)

- staattisen ja virtautetun tiedon jakeluun tarkoitettu Amazon CloudFront [7, s. 41–42]
- sovelluksille laskentakapasiteettia tarjoava Amazon Elastic Compute Cloud (EC2)
- yksinkertainen ei-relaatiotietokanta Amazon SimpleDB [7, s. 1–8].

Amazonin pilvipalvelualustan vahvuutena ovat sen tarjoamat erittäin monipuoliset ominaisuudet. Käyttäjälle annetaan myös hyvät mahdollisuudet vaikuttaa alustan toimintaan, kuten esimerkiksi käytettävään käyttöjärjestelmään. Haittapuolena on, että AWS:n koostuessa monesta erillisestä osasta joutuu osien hallintaan käyttämään enemmän resursseja. [8, s. 7–9.]

Google App Engine (GAE)

Googlen App Enginen ensimmäinen versio julkaistiin huhtikuussa 2008 [9]. GAE on tässä esitellyistä pilvipalvelualustoista yhtenäisin ja tietyssä mielessä yksinkertaisin. Yksinkertaisuus ja yhtenäisyys muodostuvat siitä, että App Engine tarjoaa vain muutamaa ydinpalvelua eikä se anna mahdollisuuksia infrastruktuuritason palveluiden, kuten käyttöjärjestelmän, muokkaamiseen. [8, s. 7–9.]

App Enginen pääkomponentit ovat seuraavat:

- Hiekkalaatikon (Sandbox) sisällä ajetaan kaikki ohjelmakoodi. Hiekkalaatikko eristää sovelluksen omaksi turvalliseksi ympäristökseen ja rajoittaa vahvasti sen pääsyä muun muassa. käyttöjärjestelmäresursseihin.
- Suoritusympäristöjen (Runtime Environment) avulla Java-, Python- ja Go-ohjelmointikielillä kehitettyjä sovelluksia voidaan ajaa. Java- ja Python-ympäristöjen tietyt ominaisuudet, kuten verkkoyhteyksien käyttö tai tiedostoihin kirjoittaminen, on estetty.
- Tallennustila (Datastore) mahdollistaa rakenteisen datan tallentamisen ja etsimisen.
- Google tilejä (Google Accounts) hyödyntämällä voidaan tunnistaa käyttäjiä App Engine -sovelluksessa.

- Ajoitetut tehtävät (Scheduled Tasks) mahdollistavat taustatehtävien suorittamisen määräajoin.
- App Engine palvelujen (App Engine Services) avulla voi mm. ladata resursseja internetistä, lähettää sähköposteja ja manipuloida kuvia. [10.]

App Enginen vahvuus on sen yksinkertaisessa käytettävyydessä, joka myös pakottaa sovelluksen rakenteen erittäin hyvin skaalautuvaksi ja käytännössä kaikki App Engineen tehdyt sovellukset skaalautuvat automaattisesti [8, s. 7–8]. Tämän lisäksi App Enginen käytön voi ainakin toistaiseksi aloittaa täysin ilman aloituskustannuksia [11]. Vastaavasti App Enginen yksinkertaisuus toimii myös sen heikkoutena rajoittaen sinne tehtäviä sovelluksia.

Windows Azure Platform

Windows Azure Platform on tässä läpikäydyistä pilvipalvelualustoista tuorein. Se esiteltiin ensi kertaa lokakuussa 2008 ja tuotantoversio julkaistiin vuoden 2009 alussa [12]. Googlen App Engineen verrattuna se tarjoaa enemmän ominaisuuksia ja paremmat mahdollisuudet infrastruktuuritason palveluiden muokkaamiseen. Amazon Web Services tarjoaa vielä enemmän ominaisuuksia ja mahdollisuuksia muokata infrastruktuuritason palveluita. Windows Azure Platformin tekniset ominaisuudet esitellään tarkemmin seuraavassa pääluvussa.

2.4 Palvelualustan valinta

Pilvipalvelualustan valintaan vaikuttavia ominaisuuksia on useita. Taulukossa 1 on verrattu Amazon Web Servicen, Google App Enginen ja Windows Azure Platformin ominaisuuksia keskenään. Lisäksi taulukossa on korostettu ominaisuuksia, jotka ovat vahvimmin tukeneet työssä käytetyn Windows Azure Platformin valintaa.

Taulukko 1. Vertailu pilvipalvelualustojen ominaisuuksista.

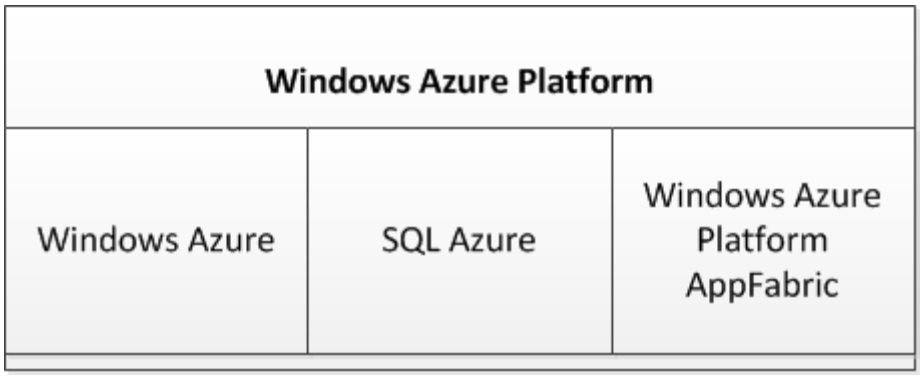
Ominaisuus	AWS	GAE	Azure Platform
Hinnoittelu	Riippuu toteutuksesta	Riippuu toteutuksesta, Voi kokeilla ilmaiseksi	Riippuu toteutuksesta
Palveluntarjoaja	Amazon	Google	Microsoft *
Ohjelmointikielet	Lähes kaikki	Java, Python ja Go.	Lähes kaikki, Pääpaino .Net-teknologian kielissä *
Muokattavuus	Erittäin kattava	Rajallinen	Kattava
Skaalautuvuus	Riippuu toteutuksesta	Hyvä	Melko hyvä
Relaatiotietokanta	Kyllä	Ei	Kyllä

** Merkittävä vaikutus työssä käytetyn alustan valintaan.*

Hinnoittelulla tarkoitetaan alustan käytöstä aiheutuvia kuluja. Vertailtujen alustojen välillä hinnoittelu on melko yhteneväinen sillä erotuksella, että Google App Engineä voi kokeilla täysin ilmaiseksi. Palveluntarjoajan merkitys korostuu, jos käytetään kyseisen palveluntarjoajan muita tuotteita. Käytettävillä ohjelmointikielillä on myös suuri merkitys. Azuren vahvuutena on erittäin hyvä tuki .Net-teknologian kielille. Pilvipalvelualustan muokattavuudella tarkoitetaan mahdollisuutta vaikuttaa esimerkiksi käyttöjärjestelmien ominaisuuksiin. Erityisesti vanhoja sovelluksia pilvipalvelualustoille siirrettäessä tällä saattaa olla suurikin merkitys. Skaalautuvuus on yksi pilvijärjestelmien kulmakivistä. Amazon Web Servicen tapauksessa skaalautuvuuden varmistamiseksi joudutaan näkemään eniten vaivaa. Relaatiotietokanta on myös tärkeä ominaisuus, jonka puuttuminen saattaa ratkaista alustan valinnan. Koska Windows Azuren ominaisuudet katsottiin yleisellä tasolla riittäviksi, nousi merkittävimmäksi tekijäksi pilvipalvelualustan valinnalle ohjelmistoyrityksen sitoutuneisuus Microsoft-teknologioihin.

3 Windows Azure Platform -pilvipalvelualusta

Windows Azure Platform on Microsoftin pilvipalvelualusta, ja se sisältää Microsoftin tarjooman pilveen tehtävää sovelluskehitystä varten [2, s. 9]. Alusta koostuu tällä hetkellä muutamasta pääosasta, ja se sijoittuu SPI-mallissa vahvimmin PaaS-tasolle, vaikkakin sen infrastruktuuriominaisuuksiin voidaan hieman vaikuttaa [13, s. 9]. Tulevissa aliluvuissa tarkastellaan alustan pääosia, joita ovat pilvikäyttöjärjestelmä Windows Azure tallennusratkaisuineen, pilvipohjainen relaatiotietokanta Sql Azure ja eri järjestelmien yhdistämiseen käytetty integraatioväylä AppFabric. Kuvio 2 esittelee alustan rakenteen sen pääosien avulla.



Kuvio 2. Windows Azure Platformin pääosat [2, s. 9–10].

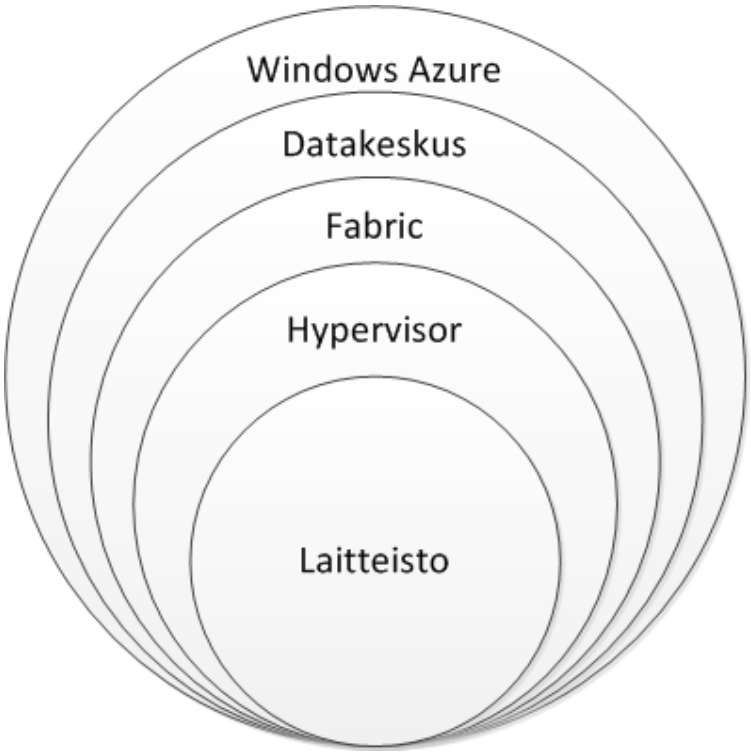
3.1 Windows Azure pilvikäyttöjärjestelmä

Windows Azure Platformin tärkein komponentti on Windows Azuren nimellä kulkeva niin sanottu ”pilvikäyttöjärjestelmä”. Windows Azuren tehtävänä on toimia alustana käyttäjän tekemille sovelluksille, sekä sen sisällä tarjottaville palveluille, kuten erilaisille tallennusratkaisuille. [13, s. 16; 14, s. 56–57.] Termillä pilvikäyttöjärjestelmä viitataan siihen, että se pyrkii tarjoamaan sovelluksille vastaavia palveluita kuin mitä perinteinen käyttöjärjestelmä on tarjonnut [2, s. 24–25]. Käyttöjärjestelmän tehtävänä on tarjota yksinkertainen rajapinta, jonka avulla eri sovellukset voivat hyödyntää tietokoneen laitteistoresursseja, kuten muistia tai suoritinta. Windows Azuren tapauksessa käytettävät laitteistoresurssit ja niiden rakenne eroavat huomattavasti perinteisestä yksittäisestä tietokoneesta, mutta tarkoituksena on edelleen yksinkertaistaa ja mahdollistaa laitteistoresurssien käyttö. [13, s. 16–17.]

Arkkitehtuuri

Windows Azure on monikerroksinen järjestelmä, joka koostuu erilaisista laitteistoista ja ohjelmistoista. Järjestelmän pohjana toimii erittäin suuri määrä palvelimia, jotka on jaettu eri puolilla maailmaa sijaitseviin datakeskuksiin. [2, s. 23–24.] Jokaiselle yksittäiselle palvelimelle on asennettu Hypervisor-ohjelmisto, jonka avulla yhdellä palvelimella voidaan ajaa useita virtuaalisia käyttöjärjestelmiä. Virtuaalisia käyttöjärjestelmiä käytetään, koska yksittäisenkin palvelimen resursseja pitää voida jakaa riittävän pienissä paloissa pilvikäyttöjärjestelmän käyttäjille. Lisäksi virtualisointi mahdollistaa sovellusten siirtämisen esimerkiksi suorituskykyisemmälle palvelimelle vain siirtämällä virtuaalikoneen palvelimelta toiselle. [2, s. 27–33.]

Kaikki yhden datakeskuksen palvelimet toimivat Fabric-ohjelmiston alaisuudessa. Sen tehtävänä on hallinnoida palvelinten laitteistoja, käyttöjärjestelmiä ja palvelimella ajettavia sovelluksia. [2, s. 23.] Lopputuloksena Fabric abstrahoi käytettävissä olevat laitteisto- ja ohjelmistoresurssit siten, että niitä voidaan helposti käyttää yhtenä lähes äärettömän suurena resurssina. [2, s. 35.] Fabric-ohjelmiston tärkein osa on muutamalla koneella ajettava Fabric Controller. Fabric Controller toimii datakeskuksen Fabric-järjestelmän aivoina ja ”omistaa” datakeskuksen kaikki laitteet koneista kytkimiin ja pystyy vaikuttamaan niiden tilaan ohjelmallisesti. Koska Fabric Controller on erittäin kriittinen osa järjestelmän toiminnalle, on jokaisesta Fabric Controllerista aina useita kopioita, jotka voidaan ottaa välittömästi käyttöön vikatilanteessa. [2, s. 35–36.] Järjestelmän arkkitehtuuria voidaan havainnollistaa Kuvion 3 mukaisella kerroksellisella rakenteella.

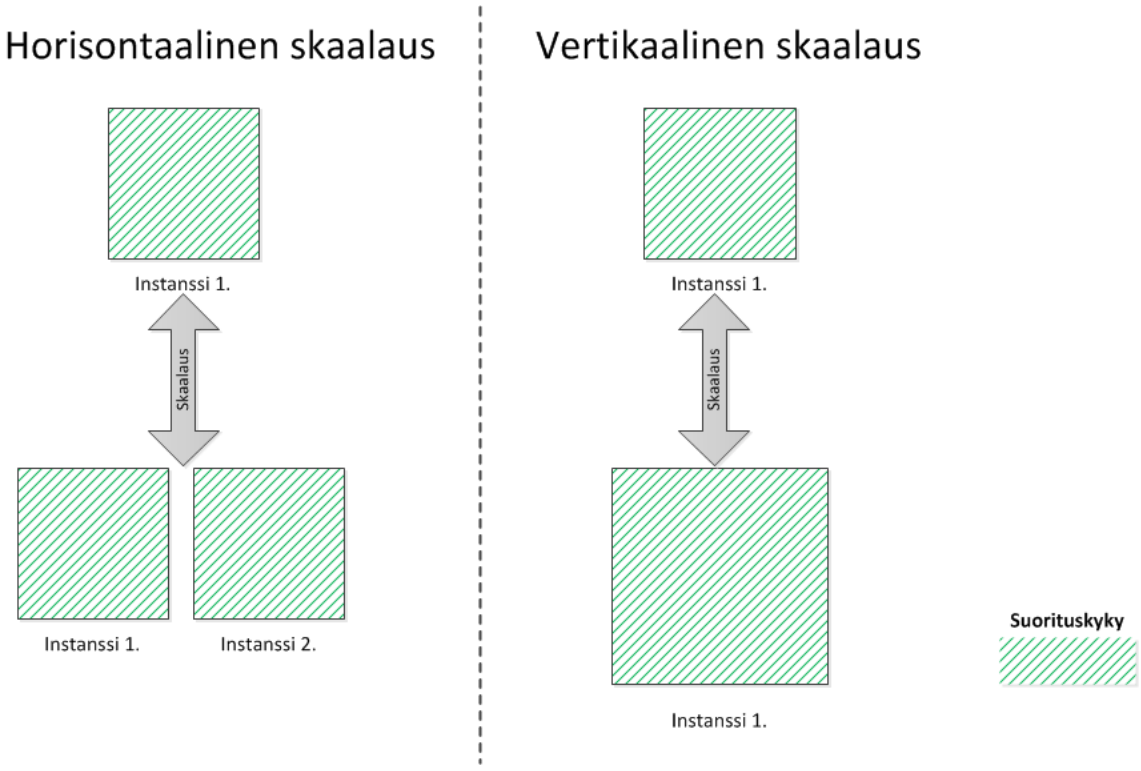


Kuvio 3. Windows Azure -arkkitehtuuri [2, s. 24].

Sovellusten ajaminen Windows Azuressa on toteutettu roolien ja rooli-ilmentymien eli rooli-instanssien avulla. Rooli on eräänlainen pohjapiirustus sovellukselle ja asettaa tiettyjä reunaehtoja sen toiminnalle. Rooli- eli sovellusinstanssi on tämän pohjapiirustuksen perusteella luotu sovellus. Jokaisella Windows Azureen asennettavalla sovelluksella tulee olla rooli ja haluttu määrä tämän roolin instansseja. Tämän lisäksi roolille määritellään roolikoko, joka vaikuttaa rooli-instanssin suorituskykyyn. Kun sovellus käynnistetään, luodaan rooliin pohjautuvia rooli-instansseja jotka ajavat sovellusta. [2, s. 67–71.]

Teknisesti rooli on itse asiassa valmis pohja, josta voidaan luoda tietyillä ominaisuuksilla varustettuja virtuaalikoneita. Rooli-instanssi on tästä pohjasta luotu virtuaalikone, johon on asennettu ajettava sovellus. Jokainen sovellusinstanssi ajetaan siis omassa virtuaalikoneessaan. [2, s. 67–71; 14, s. 8.] Roolikoko vaikuttaa virtuaalikoneen muistin, prosessoritehon ja väliaikaisen tallennustilan määrään sekä tiedonsiirtonopeuteen näiden komponenttien välillä. [2, s. 72.] Koska yhdestä sovelluksesta voidaan luoda useita sovellusinstansseja, on sovellus skaalattavissa lisäämällä tai vähentämällä sovellusinstanssien määrää. Tätä tapaa, jossa suorituskykyä muutetaan vaihtamalla proses-

soivien osien lukumäärää, kutsutaan horisontaaliseksi skaalaamiseksi. Vastaavasti yhden prosessoivan osan kapasiteetin muutos, eli tässä tapauksessa roolikoon muuttaminen, on vertikaalista skaalaamista [4, s. 258]. Kuviossa 4 on esitetty visuaalisessa muodossa horisontaalisen ja vertikaalisen skaalauksen erot.

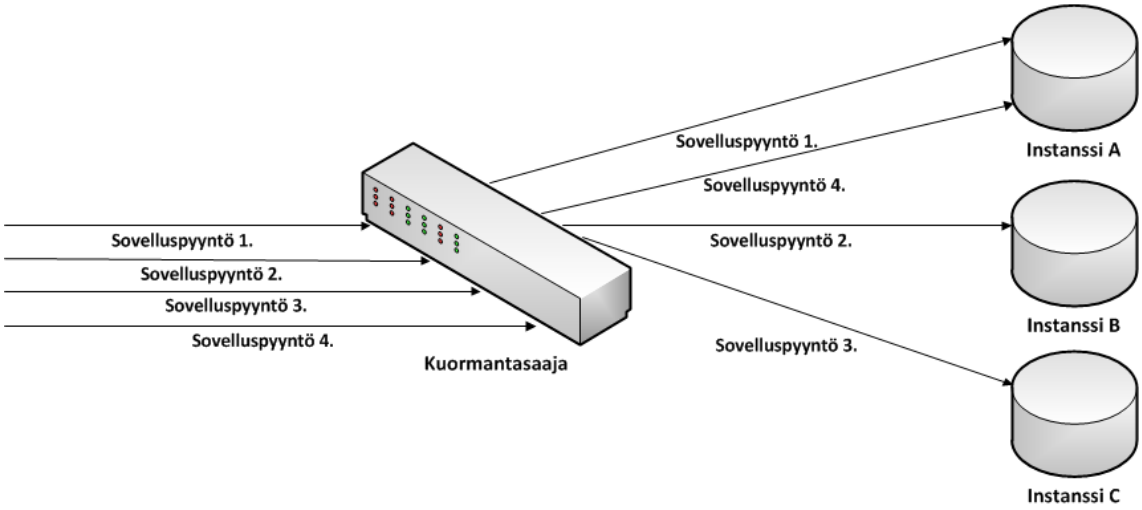


Kuvio 4. Horisontaalinen ja vertikaalinen skaalaus Windows Azuressa.

Kuviosta voi huomata, että palvelun suorituskykyä voidaan lisätä kasvattamalla roolikokoa (vertikaalinen skaalaus) tai lisäämällä instanssien määrää (horisontaalinen skaalaus). Molemmat skaalaustavat kasvattavat kuvion tapauksessa suorituskykyä yhtä paljon. Horisontaalisen skaalauksen etuna on kuitenkin se, että sitä käytettäessä suorituskykyä voidaan kasvattaa lähes rajattomasti. Vertikaalista skaalausta käytettäessä tulee vääjäämättä raja vastaan, jolloin ei enää voida kasvattaa yksittäisen instanssin suorituskykyä lisäämällä esimerkiksi prosessoritehoa.

Windows Azure pystyy hyödyntämään useita sovellusinstansseja Fabricin hallinnoimien kuormantasaajien avulla. Kun sovellukseen saapuu sovelluspyyntö, kuormantasaaja ottaa pyynnön vastaan. Kuormantasaaja tuntee kaikki sovelluksen sovellusinstanssit ja osaa jakaa saapuvat sovelluspyynnöt tasaisesti niiden kesken. Jos siis sovelluksella on

kolme sovellusinstanssia ja sinne saapuu kuusi pyyntöä, välitetään jokaiselle sovellusinstanssille kaksi sovelluspyyntöä. [2, s. 69–70.] Kuvio 5 havainnollistaa kuormantasausta.



Kuvio 5. Kuormantasaus Windows Azuressa.

Kuviossa palveluun saapuu neljä sovelluspyyntöä, jotka kuormantasaaja jakaa instanssien kesken. Koska käytössä on ainoastaan kolme instanssia, välitetään yhdelle niistä kaksi sovelluspyyntöä.

Windows Azureen voidaan kehittää sovelluksia useilla eri ohjelmointikielillä [14, s. 139]. Koska kyseessä on Microsoftin tuote, on pääpaino Microsoftin .Net -teknologian kielissä joille on olemassa myös eniten dokumentaatiota ja tukea. Microsoft on kehittänyt valmiita sovelluskomponentteja useille ohjelmointikielille, jotta sovelluskehitys ja alustan käyttöönotto helpottuisivat [14, s. 139].

Tällä hetkellä järjestelmä tukee muun muassa seuraavia ohjelmointikieliä:

- .Net(C#, Visual Basic ja F#)
- C++
- PHP
- Ruby
- Java
- Python [14, s. 139].

Web-Rooli

Windows Azuren web-rooli on tarkoitettu alustaksi web-sovelluksille. Web-sovelluksella tarkoitetaan sovellusta, johon käyttäjä ottaa yhteyttä http- tai https-protokollaa käyttäen. Tyypillisiä web-sovelluksia ovat erilaiset WWW-sivut ja WWW-sovelluspalvelut. Web-roolissa toimiva sovellus ajetaan aina Microsoftin WWW-sovelluspalvelimen Internet Information Services 7:n (IIS) sisällä. [2, s. 68.]

Internet Information Services on hyvin yleinen www-sovelluspalvelin, ja lähes kaikkia Microsoftin teknologiaan pohjautuvia web-sovelluksia käytetään IIS:n sisällä. Tästä johtuen web-sovelluksen ajaminen Azuressa on hyvin samankaltaista kuin perinteisessä palvelinympäristössä. [14, s. 84.] IIS:n avulla web-rooli mahdollistaa myös muiden kuin Microsoft ohjelmointikielien, kuten php:n, käytön WWW-sovelluskehityksessä. Web-roolin suurimpana rajoitteena on, että se tukee ainoastaan http- ja https-protokollia sisään ja ulos kulkevalle liikenteelle [15, s. xix].

Työrooli

Työrooli on geneerinen rooli, jota ei ole valmiiksi suunniteltu tiettyä tehtävää varten toisin kuin web-roolia. Työroolin perusideana on se, että pilvikäyttöjärjestelmä käynnistää siitä luodun instanssin ja tämä instanssi jää suorittamaan siihen ohjelmoitua toiminnallisuutta niin pitkään kun se on tarpeellista. [2, s. 84.] Ehkä tärkein tehtävä työroolille on erilaisten viestijonojen purkaminen. Viestijonojen käyttö mahdollistaa sovelluksen eri osien erottamisen toisistaan niin, että sovellusta voidaan skaalata osissa joko muuttamalla viestien vastaanottokapasiteettia tai viestien käsittelykapasiteettia. Tyypillisesti viestien vastaanottoon käytetään web-roolia ja viestien käsittelyyn työroolia. [13, s. 31.]

Esimerkkinä työroolin ja viestijonon käytöstä voisi olla vaikkapa verkkokauppa, jossa asiakkaan tehdessä tilauksen tallennetaan jonoon viesti ”muodosta lasku”, joka sisältää tilauksen tiedot. Kyseisessä jonossa olevat viestit voidaan lukea työroolin avulla ajastetusti kerran päivässä ja suorittaa laskujen luonti. Näin pystytään jakamaan sovellukselle aiheutuvaa kuormaa sellaisiin ajankohtiin, jotka ovat sen kannalta sopivimpia.

Muita tyypillisiä käyttökohteita työroolille ovat muun muassa erilaiset eräajot, joissa sovellus suorittaa jonkun tietyn tehtävän esimerkiksi kerran päivässä tiettyyn aikaan, tai pitkäkestoiset laskentatehtävät kuten kuvankäsittely. Työrooli mahdollistaa myös muidenkin kuin http- ja https-protokollien käytön sisään ja ulos tulevalle liikenteelle [14, s. 338–339.] Näin työroolia voidaan käyttää esimerkiksi suorituskykyistä tcp-protokollaa hyödyntävän verkkomoninpelipalvelimen rakentamiseen.

Virtuaalikonerooli

Virtuaalikonerooli mahdollistaa oman virtuaalikoneen asentamisen roolina. Käytännössä jo olemassa olevan Windows Server 2008 R2 -käyttöjärjestelmän ja siinä toimivat sovellukset voi siirtää lähes sellaisenaan suoraan pilvialustalle. Ratkaisu tarjoaa reilusti joustavuutta, koska käyttöjärjestelmän konfigurointi on käyttäjän tehtävänä. Virtuaalikonerooli on erityisen nopea tapa siirtää sovellus pilvialustalle. [13, s. 32.] Skaalautuvuuden kannalta tämä ei kuitenkaan välttämättä ole paras ratkaisu, sillä jo olemassa olevia sovelluksia ei aina ole suunniteltu skaalautuviksi eivätkä ne hyödynnä Windows Azuren skaalautuvia ominaisuuksia.

3.2 Pilvikäyttöjärjestelmän tallennusratkaisut

Windows Azure tarjoaa neljää erityyppistä tallennusratkaisua sovellusten käytettäväksi. Nämä tallennusratkaisut ovat binääriobjektit (blob), taulut (table), jonot (queue) ja paikallinen tallennustila (local storage). [13, s. 79–135.] Tallennusratkaisuista binääriobjektit, taulut ja jono ovat erittäin hyvin skaalautuvia. Niitä käytetään rest-tyyppisen http-rajapinnan avulla (RESTful http API). Rest-tyyppisen rajapinnan käyttö tarkoittaa sitä, että tallennusjärjestelmää käytetään yksinkertaisten http-pyyntöjen avulla. Tässä tapauksessa rajapinta toimii siten, että http-pyyntöjä url-osoitteisiin voidaan pitää operaatioina, joita kyseinen url-osoite suorittaa. Erilaisten http-pyyntöjen avulla tallennusjärjestelmästä voidaan hakea tietoja, sinne voidaan lisätä tietoja ja sieltä voidaan poistaa tietoja. Tallennusjärjestelmiä voidaan käyttää sekä pilvikäyttöjärjestelmän sisäpuolelta esimerkiksi www-roolista, että myös sen ulkopuolelta, kuten vaikka mobiilisovel-

luksesta. Koska lähes kaikilla ohjelmointikielillä ja alustoilla on mahdollista tehdä http-pyyntöjä, on Windows Azuren tallennusratkaisuja mahdollista käyttää lähes mistä vain edellyttäen, että verkkoyhteys internetiin on käytettävissä. [2, s. 127–139.] Rest-tyyppisten tallennusratkaisujen pohjana toimii Windows Azure Platformiin luotava tallennustili (storage accaunt). Tallennustilin sisälle on mahdollista luoda haluttu määrä binääriobjekteja, tauluja ja jonoja.

Binääriobjekti eli blob (binary large object) on tallennusratkaisu binäärimuotoisen tiedon tallentamiseen. Blobin käyttötarkoitus on pitkälle sama kun perinteisen levyjärjestelmän, eli tarjota tiedostoille tallennuspaikka. Blobiin voidaan tallentaa mitä tahansa tiedostoja, kuten kuvia, videoita ja tekstiä. Blobin datamalli muodostuu säiliöistä (containers) ja niissä sijaitsevista yksittäisistä tiedostoista eli blobeista. Yhdessä blobissa voi itse sisällön lisäksi olla myös kahdeksan kilotavua metadataa, jolla voidaan kuvata sen sisältöä. [2, s. 157–162.]

Taulut mahdollistavat taulukkomuotoisen datan tallentamisen [2, s. 226]. Vaikka termi taulu viittaa relaatiotietokantoihin, on niillä merkittäviä eroavaisuuksia. Windows Azuren taulut eivät sisällä viite-avaimia eivätkä siten myöskään tue viite-eheyttä, jota pidetään relaatiotietokannan perusominaisuutena. Tauluihin ei myöskään ole valmista kyselykieltä, jonka avulla voitaisiin linkittää eri taulujen rivejä toisiinsa. Toistaiseksi taulut eivät myöskään tue valinnaisia indeksejä, joiden avulla voitaisiin nopeuttaa tietojen hakua yksittäisten sarakkeiden perusteella. [14, s. 239–241.]

Taulutallennusratkaisu voidaan kuvata kolmella osalla:

- entiteetillä, joka kuvastaa taulun yhtä riviä
- ominaisuudella, joka kuvastaa entiteetin yhtä ominaisuutta
- taululla, joka sisältää kokoelman entiteettejä. [2, s. 227]

Taulun entiteeteille voidaan lisätä dynaamisesti tarvittava määrä eri ominaisuuksia ja näin tauluilla voidaan mallintaa haluttuja tietorakenteita. Yksi entiteetti voi kuitenkin maksimissaan sisältää yhden megatavun dataa. [2, s. 226–229.] Jokaisella taulun entiteetillä on kolme pakollista ominaisuutta: osioavain eli ParititionKey, riviavain RowKey ja aikaleima TimeStamp. Osioavain määrittää, mihin osioon kyseinen entiteetti kuuluu. Riviavaimen avulla entiteetti tunnistetaan osion sisällä ja aikaleima kertoo, milloin enti-

teettiä on viimeksi muokattu. Entiteetin yksilöllinen tunniste muodostuu osioavaimen ja riviavaimen yhdistelmästä. [14, s. 224–259.] Tätä voidaan verrata reaalimaailman puhelinnumeroon, jossa osioavain kuvaa maakoodia ja riviavain itse numeroa. Osio- ja riviavaimen käyttö liittyy taulujärjestelmän skaalautuvuuteen. Tietomäärän kasvaessa tulee ennen pitkään vastaan tilanne, jossa kaikkea yhteenkuuluvaa tietoa ei yksinkertaisesti voida tallentaa yhdelle palvelimelle. Tästä syystä Windows Azure voi jakaa yhden taulun dataa usealle fyysiselle palvelimelle. Tämä datan jakaminen toteutetaan osioavaimen avulla. Kaikki taulun entiteetit, joilla on sama osioavain sijaitsevat samalla palvelimella. Vastaavasti eri osioavaimilla olevat entiteetit voidaan sijoittaa eri palvelimille tarpeen vaatiessa. [2, s. 248–254.] Taulukko 2 havainnollistaa yhden taulun rakennetta ja sisältöä.

Taulukko 2. Kuvitteellisen henkilöt taulun rakenne ja sisältö.

Osiovain	Riviavain	Aikaleima	Etunimi	Sukunimi
A	1	20110312160342	Ville	Ahonen
A	2	20110312160353	Simo	Aalto
K	1	20110312160302	Sini	Keränen
K	2	20110312160311	Lasse	Kokkonen

- Palvelin 1. Entiteetit osioavaimella A
- Palvelin 2. Entiteetit osioavaimella K

Tauluun on tallennettu henkilöitä, joille on annettu pakollisten ominaisuuksien lisäksi Etunimi- ja Sukunimi-ominaisuudet. Taulun entiteettien osioavaimeksi on valittu sukunimen ensimmäinen kirjain. Tällä tavoin pilvikäyttöjärjestelmä pystyy jakamaan entiteetit eri palvelimille.

Tauluja käyttöönotettaessa on syytä kiinnittää huomiota osioavaimeen ja varmistaa, että sen avulla voidaan jakaa data järkevästi. Osioavain on myös erittäin tärkeä, kun entiteettejä etsitään taulusta. Entiteettejä voidaan etsiä taulusta minkä tahansa ominaisuuden perusteella. Hakeminen on kuitenkin huomattavasti suorituskykyisempää, jos hakuehdoksi saadaan myös osioavain mahdollisten muiden ominaisuuksien lisäksi. [2, s. 251–255.]

Windows Azuren kolmas rest-pohjainen tallennusratkaisu on jono. Se mahdollistaa viestien tallentamisen ja lukemisen siinä järjestyksessä, jossa ne on tallennettu. Tieto-

jenkäsittelyssä jono on rakenne, joka toimii niin sanotulla first in first out (fifo) -periaatteella. Fifo-periaate tarkoittaa sitä, että jonoon voidaan laittaa viestejä ja kun jono halutaan purkaa, se tehdään aloittaen jonossa pisimpään olleesta viestistä. [14, s. 357–359.] Tämä yksinkertainen periaate mahdollistaa erilaisten asynkronisten prosessien toteuttamisen. Kun jonoon laitetaan viesti, ei viestin laittaja tiedä milloin viesti käsitellään. Viestin laittaja joutuu sopeuttamaan toimintansa sen mukaan, että viesti käsitellään ”joskus”. Tämä johtaa siihen, että viestin laittaja ja sen käsittelijä ovat erittäin löyhästi sidoksissa toisiinsa. Tällöin prosessin kulkua voidaan tarvittaessa skaalata molemmista päistä muuttamalla käsittelijöiden tai viestin laittajien lukumäärää. [2, s. 204–209.] Arkielämässä vastaava tilanne tulee vastaan esimerkiksi ravintolassa, jossa palvelua voidaan skaalata muuttamalla kassavirkailijoiden ja kokkien lukumääriä tarpeen mukaan.

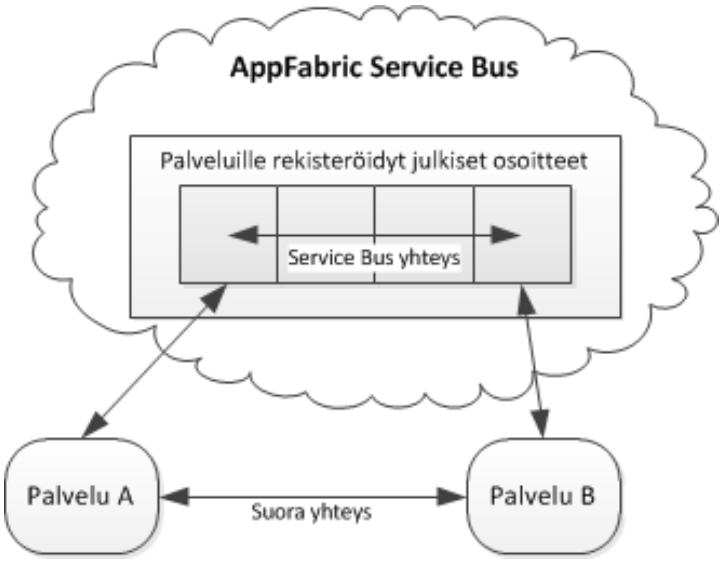
Windows Azuressa jonoja käytetään tyypillisesti tilanteessa, jossa WWW-rooli-instanssi ottaa vastaan viestin ja laittaa sen jonoon. Tämän jälkeen työrooli-instanssi käy jonoa läpi tietyin väliajoin. [13, s. 128.] Tällaista järjestelyä voidaan soveltaa esimerkiksi verkkokaupan tilaustenkäsittelyssä seuraavasti. Verkkokaupan käyttäjä selailee WWW-roolissa toimivaa sivustoa ja täyttää siellä tilauksen. WWW-rooli ottaa tilauksen vastaan ja laittaa jonoon viestin ”tulosta lasku tilaukselle”. Verkkokauppaan tehtyjen tilausten laskut tulostetaan ajastetusti esimerkiksi kerran päivässä. Tällöin työrooli purkaa jonon ja tulostaa jonosta löytyvät laskut paperille.

Paikallinen tallennustila määritellään roolikohtaisesti käyttöön, ja se vastaa toiminnaltaan lähes normaalia levyjärjestelmää. Yksittäinen rooli-instanssi saa käyttöönsä roolissa määritellyn määrän tallennustilaa. Jokaisella rooli-instanssilla on täysin oma tallennustilansa. Ne eivät siis voi jakaa yhteistä paikallista tallennustilaa. Tästä syystä paikallisen tallennustilan käyttötapaukset ovat melko rajallisia. Se on ensisijassa suunniteltu tiedon väliaikaiseen tallentamiseen. Koska paikallinen tallennustila sijaitsee rooli-instanssissa, sitä voidaan käyttää erittäin nopeasti verrattuna esimerkiksi blob-tallennustilaan, jota joudutaan käyttämään rest-kutsujen avulla. [13, s. 80–85.]

Liite 1 tarjoaa kokonaiskuvan Windows Azuren tallennusratkaisuista. Liitteessä on alhaalla kuvattuna rooli-instanssit, joilla on oma paikallinen tallennustilansa. Rooli-instanssit käyttävät tallennustilien sisällä olevia binääriobjekteja, tauluja ja jonoja.

3.3 Integraatioväylä Azure AppFabric

Usein sovellukset koostuvat monista yksittäisistä palveluista, jotka sijaitsevat eri paikoissa ja joita on toteutettu eri teknologioilla. Azure AppFabric on suunniteltu helpottamaan eri järjestelmien välisiä integraatioita. AppFabric koostuu tällä hetkellä kahdesta pääosasta: palveluväylästä (Service Bus) ja pääsynhallintapalvelusta (Access Control Service) [14, s. 379–381]. Palveluväylän avulla voidaan yhdistää eri palveluita. Sen päätehtävänä on mahdollistaa kommunikointi eri järjestelmien välillä, vaikka ne sijaitsisivat palomuurien takana tai vaihtaisivat sijaintiaan säännöllisesti. Järjestelmiä voidaan yhdistää toisiinsa rekisteröimällä ne väylän tarjoamiin julkisiin osoitteisiin ja sallimalla sovelluksille yhteys väylään. Rekisteröinnin jälkeen palvelua voidaan käyttää väylän tarjoamassa osoitteessa. Tällöin viestiliikenne kohdistetaan väylän tarjoamaan julkiseen osoitteeseen, josta se kulkee väylän läpi itse palveluun. [14, s. 397–401.] Tarvittaessa palveluväylä voi myös ilmoittaa sinne rekisteröidyn palvelun osoitteen, jos palveluun halutaan ottaa yhteys, joka ei mene väylän läpi. [13, s. 182.] Kuvio 6 havainnollistaa palveluväylän toimintaa palveluita yhdistettäessä.

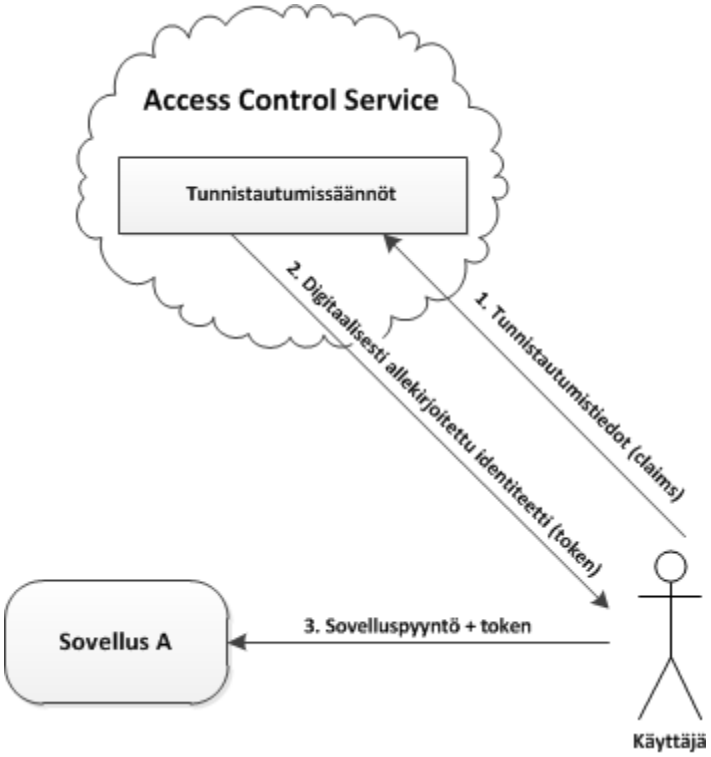


Kuvio 6. AppFabric Service Bus [14, s. 399; 16].

Kuviossa kaksi palvelua on rekisteröity palveluväylään. Se tarjoaa palveluille julkiset osoitteet joiden, kautta ne voivat kommunikoida keskenään väylän välityksellä. Lisäksi

palveluväylä voi kertoa palveluiden suorat osoitteet, jolloin palvelut voivat muodostaa kuviossa näkyvän suoran yhteyden.

Access Control Service (ACS) on palvelu, jonka avulla sovellusten ja palveluiden käyttäjille voidaan myöntää käyttöoikeuksia haluttujen sääntöjen perusteella. Palvelu vastaanottaa käyttäjän lähettämiä tunnistautumistietoja (claims), joita voivat olla esimerkiksi käyttäjätunnus ja salasana. Näitä tietoja ACS vertaa sinne määriteltyihin tunnistautumissääntöihin. Sääntöjen perusteella ACS luo digitaalisesti allekirjoitetun identiteetin eli tokenin ja palauttaa tokenin tunnistautuvalle osapuolelle. Tämän jälkeen tunnistautuva osapuoli käyttää tokenia tunnistautuessaan varsinaiselle sovellukselle, jota se haluaa käyttää. Sovellus tutkii tokenin ja päättää, mitä toimia sen pohjalta sallitaan. [14, s. 381–396.] Kuvio 7 esittelee ACS:n käyttöä.



Kuvio 7. Access Control Servicen käyttö. [14, s. 391]

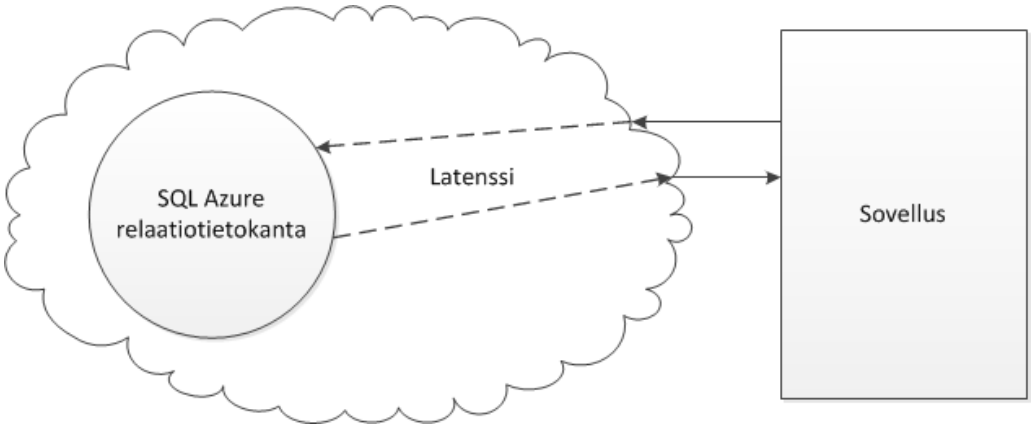
Kuviossa käyttäjä haluaa tehdä pyynnön sovellukselle A. Jotta tämä onnistuisi, tulee käyttäjän ensin lähettää tunnstautautumistietoja Access Control Serviceen. ACS vertaa tunnistautumistietoja tunnustautumissääntöihin ja luo niiden perusteella käyttäjälle tokenin. Nyt käyttäjä voi tunnistautua liittämällä tokenin sovelluspyyntöön.

3.4 Pilvitietokanta SQL Azure

Vaikka Windows Azure sisältää useita tallennusratkaisuja, tarvitsevat useat sovellukset silti relaatiotietokantaa. Tätä varten Windows Azure Platform sisältää pilvipohjaisen relaatiotietokannan SQL Azuren. Vaikka monissa tapauksissa jo olemassa oleva relaatiomalli olisikin muunnettavissa käyttämään Windows Azuren tauluja, on silti relaatiotietokannalla tiettyjä etuja. Näitä ovat mm. viite-eheys, kehittäjille jo ennestään tuttu sql-kyselykieli sekä nopeus. Näiden lisäksi vanhojen sovellusten siirtäminen helpottuu ja nopeutuu, kun relaatiotietokanta voidaan siirtää lähes suoraan pilvialustalle.

SQL Azure on hyvin pitkälle yhteneväinen Microsoftin relaatiotietokantatuotteen SQL Serverin kanssa. Suurimpina eroina SQL Server -tietokantaan on puuttuva tekstihaku sekä puuttuva mahdollisuus ajaa ohjelmakoodia tietokannan sisällä [13, s. 252]. SQL Azure -tietokannoille on myös asetettu 50 gigatavun maksimikokorajoitus [13, s. 243]. Kokorajoitus ei kuitenkaan muodostu merkittäväksi ongelmaksi, sillä tietokantojen kasvaessa näin suuriksi joudutaan tietoja kuitenkin jakamaan useiden tietokantojen kesken.

Merkittävä ero perinteisen ja pilvipohjaisen relaatiotietokannan välillä syntyy verkkolatenssista. Latenssilla tarkoitetaan aikaa, joka kuluu tiedonsiirrossa käytetyiltä paketeilta kulkea lähettäjältä vastaanottajalle. Suurempi latenssi hidastaa sovelluksen toimintaa. SQL Azuren latenssi on useimmiten suurempi kuin perinteisen relaatiotietokannan. Tämä johtuu siitä, että pilvipohjainen ratkaisu sijaitsee todennäköisesti eri paikassa kuin sitä käyttävä sovellus. Lisäksi pilviratkaisuissa relaatiotietokannan fyysinen paikka saattaa vaihdella, ja näin syntyy myös vaihtelua verkkolatenssiin. Latenssin aiheuttamia hidastavia vaikutuksia voidaan minimoida muutamin keinoin. Yhteyden muodostaminen tietokantaan on hidasta, tätä tulisikin vältää uudelleenkäyttämällä sovelluksessa jo luotuja yhteyksiä mahdollisimman paljon. Lisäksi tietokantaan tehtävien kyselyjen määrää on hyvä minimoida suosimalla muutamia suuria kyselyitä useiden pienten sijaan. [2, s. 335–336.] Kuvio 8 havainnollistaa verkkolatenssia SQL Azuressa.



Kuvio 8. Latenssi SQL Azure relaatiotietokantaa käytettäessä.

Kuviosta voidaan havaita, että tietokannan sijaitessa pilvessä ei voida tarkalleen tietää kuinka suureksi latenssi tietokannan ja sovelluksen välillä muodostuu. Useimmiten se on kuitenkin suurempi verrattuna tilanteeseen, jossa sovellus ja tietokanta ovat samassa konessalissa tai jopa samalla palvelimella. Lisäksi latenssi saattaa vaihdella suurestikin, mikä johtuu pilvijärjestelmän tavasta optimoida omaa toimintaansa.

3.5 Hinnoittelu

Windows Azure Platformin eri osat on hinnoiteltu pääosin käyttömääräperusteisesti. Lisäksi tarjolla on erilaisia valmispaketteja, joihin on valmiiksi sisällytetty tietty kiinteä käyttömäärä. Hinnoittelumalli on hyvin samankaltainen kuin matkapuhelinliittymissä, joissa voidaan käyttömääräperusteisen hinnoittelun lisäksi valita erilaisia puhe -ja tekstiviestipaketteja. Hinnoitteluperusteena käytetyllä transaktiolla tarkoitetaan kirjoitus -ja lukuoperaatioita kyseiseen palveluun. Palveluiden hinnoittelu on sidottu dollariin, joten dollarin kurssi vaikuttaa hintatasoon. Taulukko 3 sisältää esimerkkejä pilvipalvelualustan käyttömääräperusteisista hinnoista.

Taulukko 3. Esimerkkejä Windows Azure Platformin käyttömääräperusteisesta hinnoittelusta [17].

Windows Azure -roolit	Dollaria
Erittäin pieni instanssi *	0,05
Pieni instanssi *	0,12
Keskikokoinen instanssi *	0,24
Suuri instanssi *	0,48
Erittäin suuri instanssi *	0,96
Windows Azure tallennusratkaisut	
Tallennettu gigatavu **	0,15
Transaktiot 10000 kpl.	0,01
SQL Azure	
Tietokanta 1 Gt:uun asti **	9,99
Tietokanta 5 Gt:uun asti **	49,95
Tietokanta 10 Gt:uun asti **	99,99
Tietokanta 20 Gt:uun asti **	199,98
Tietokanta 30 Gt:uun asti **	299,97
Tietokanta 40 Gt:uun asti **	399,96
Tietokanta 50 Gt:uun asti **	499,95
Tiedonsiirto Euroopassa ja Pohjois-Amerikassa	
Gigatavu ulospäin **	0,15
Tiedonsiirto Aasian ja Tyynenmeren alueella	
Gigatavu ulospäin**	0,20
AppFabric	
Service Bus yhteys 1 kpl.	3,99
Access Control transaktiot 100000 kpl.	1,99

* *per tunti*
** *per kuukausi*

Taulukon hintatietojen perusteella voidaan arvioida kuukausikustannuksia kuvittelliselle WWW-sivustolle. Kuvittelliseella WWW-sivustolla

- yhden sivun lataamiseen käytetään keskimäärin 200 kilotavua ja kymmenen transaktiota
- sivusto käyttää yhden pienen sovellusinstanssin
- sivustolla on kymmenen gigatavua tallennettua dataa
- sivulatauksia tulee 3000 kappaletta kuukaudessa.

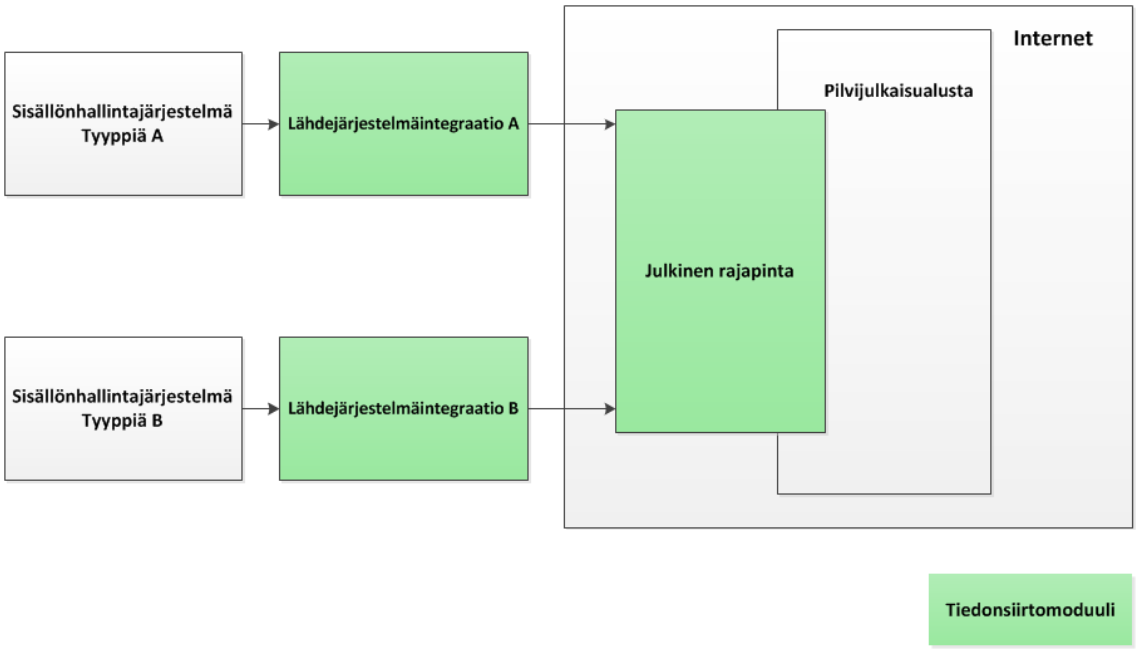
Näiden tietojen pohjalta voidaan arvioida sivuston kustannuksiksi noin 92 $ kuukaudessa. Kustannukset muodostuvat seuraavista osista.

- Tallennetut tiedot 1,5 $ (10 Gt x 0,15 $)
- Transaktiot 0,03$ (3000 x 10 / 10000 x 0,01 $)
- Tiedonsiirto 0,86 $ (3000 x 0,0019 Gt x 0,15 $)
- Sovellusinstanssi 90 $ (31 x 24 h x 0,12 $)

4 Tiedonsiirtomoduulin suunnittelu

4.1 Vaatimusmäärittely

Tarkoituksena oli kehittää internetin välityksellä käytettävä tiedonsiirtomoduuli, jonka avulla voidaan siirtää tietoa eri sisällönhallintajärjestelmistä yhteen pilvipalveluna toteutettuun julkaisualustaan. Lisäksi toteutettiin tiedonsiirtomoduulin esimerkki-integraatio yhteen sisällönhallintajärjestelmään. Moduuli toimii osana pilvijulkaisualustaa, joka tullaan toteuttamaan Windows Azure Platform -pilvipalvelualustan avulla. Pilvijulkaisualustaa kehitettiin tiedonsiirtomoduulin kanssa samaan aikaan. Tiedonsiirtomoduulilla tarkoitetaan tässä yhteydessä järjestelmää, joka tarjoaa jonkinlaisen julkisen rajapinnan toteutuksineen, protokollan, jota noudattamalla tietoa siirretään pilvijulkaisualustalle ja integraatiokomponentin lähdejärjestelmään (ks. kuvio 9.). [18.]



Kuvio 9. Yleiskuvaus tiedonsiirtomoduulista.

Tiedonsiirtomoduulille määriteltiin seraavanlaiset toiminnalliset ja laadulliset vaatimukset.

Toiminnalliset vaatimukset

- Tiedonsiirtomoduulin päätehtävänä on, että sen avulla pystytään siirtämään eri sisällönhallintajärjestelmien lähettämää tietoa pilvipalvelualustalle ja hyödyntämään sitä siellä sijaitsevassa pilvijulkaisualustassa.
- Integraatio, eri sisällönhallintajärjestelmiin tulee olla helposti toteutettavissa.
- Aluksi ohjelmisto mahdollistaa datan siirtämisen ainoastaan yhteen suuntaan eli sisällönhallintajärjestelmästä pilvijulkaisualustaan [19].
- Tiedonsiirtomoduulin tulee sisältää mekanismi, jolla sen käyttäjät voidaan tunnistaa [20].
- Ohjelmiston tulee tarvittaessa pystyä ottamaan data vastaan pienissä paloissa ja koostamaan se takaisin alkuperäiseen muotoonsa [19].

Laadulliset vaatimukset

- Ylläpidettävyys: Tiedonsiirtomoduulia tullaan jatkokehittämään usean sovelluskehittäjän toimesta. Ohjelmiston kehitys tulee siis suorittaa yleisiä ohjelmistokehityskäytäntöjä noudattaen ja koodia tulee kommentoida riittävästi. Koska ohjelmistoyritys käyttää Microsoftin .Net -teknologiaa ja ohjelmointikielenä toimii C#, tulee mahdollisimman suuri osa ohjelmoinnista tulee suorittaa kyseisellä ohjelmointikielellä.
- Tehokkuus: Pilvijulkaisualustalla ja siten myös tiedonsiirtomoduulilla tulee olemaan erittäin suuri määrä käyttäjiä. Ohjelmiston tulee toimia mahdollisimman tehokkaasti, ja sitä on pystyttävä skaalaamaan tarpeen mukaan [19].
- Vikasietoisuus: Tiedonsiirtomoduuli on erittäin kriittinen osa pilvijulkaisualustaa ja sitä käytettään julkisen rajapinnan kautta. Tästä syystä ohjelmiston tulee olla erittäin vikasietoinen eikä sen suoritus saa häiriintyä missään tapauksissa julkista rajapintaa käytettäessä.
- Testattavuus: Koko pilvijulkaisualustalle suunnitellaan automaattista testausta, joten tiedonsiirtomoduulin tulee myös olla testattavissa [19].
- Turvallisuus: Tiedonsiirtomoduuli siirtää dataa julkisen rajapinnan avulla, joten tiedonsiirron tulee olla salattua.

- Joustavuus: Tiedonsiirtomoduulin avulla tullaan siirtämään dataa erilaisista lähdejärjestelmistä. Tämä johtaa siihen, että dataa joudutaan käsittelemään lähdejärjestelmäkohtaisesti, jotta se saadaan muutettua pilvijulkaisualustalle sopivaan muotoon. Tiedonsiirtomoduulin lähdejärjestelmäriippuvaisten osien tulee olla mahdollisimman helposti jatkokehitettävissä.

4.2 Toteutusmenetelmän valinta

Ensimmäinen askel suunnittelussa oli tutkia, mitkä olisivat sopivat teknologiat, joita hyödyntämällä käytännön toteutuksesta tulisi mahdollisimman hyvä ja turhalta työltä vältyttäisiin. Tiedonsiirto järjestelmästä toiseen on hyvin yleinen tehtävä, ja sitä varten on suunniteltu useita valmiita ohjelmistoja ja ohjelmakirjastoja. Aluksi tutkittiin, mitä mahdollisuuksia valmiista ohjelmistoista ja ohjelmakirjastoista löytyisi tehtävän suorittamiseen. Alustavien tutkimusten pohjalta valikoitiin kolme mahdollista ratkaisuvaihtoehtoa, joiden soveltuvuutta päätettiin tutkia tarkemmin. Nämä vaihtoehdot olivat Microsoft Sync Framework, SQL Azure Data Sync ja kokonaan itse toteutettu ohjelmisto. SQL Azure Data Sync valikoitui vaihtoehdoksi sillä perusteella, että se on suunniteltu tiedon synkronointiin pilvipalvelualustalla. Sync Framework vaikutti melko kattavalta ja yleiskäyttöiseltä tiedonsynkronointi alustalta. Itse toteutettua ohjelmistoa pidettiin luonnollisesti yhtenä vaihtoehtona, sillä näin ohjelmiston ominaisuuksiin päästäisiin vaikuttamaan kaikista kattavimmin.

Vaatimukset asettavat tiedonsiirtomoduulille muutamia reunaehtoja. Moduuli toimii osana pilvijulkaisualustaa, joka on toteutettu Windows Azure Platform -pilvipalvelualustalle. Tästä syystä tiedonsiirtomoduulin pääosa eli julkinen rajapinta on luontevinta toteuttaa samalle pilvipalvelualustalle. Näin koko julkaisualusta pysyy aitona pilviratkaisuna. Teknisesti olisi mahdollista toteuttaa moduulin julkinen rajapinta perinteisen palvelinkeskuksessa sijaitsevan palvelimen avulla ja siirtää sieltä data pilvijulkaisualustaan. Tämä kuitenkin hankaloittaisi julkaisualustan hallintaa, jos sen tärkeä osa sijaitsisi eri ympäristössä. Lisäksi tiedonsiirtomoduulin kehittäminen pilvipalvelualustalle alustalle mahdollistaa sen helpon skaalaamisen vaatimusten mukaisesti.

Toinen reunaehto muodostuu siitä, että tiedonsiirtomoduuli tulee olla helposti integroitavissa eri sisällönhallintajärjestelmiin. Sisällönhallintajärjestelmiä on olemassa useita, ja monet niistä on kirjoitettu eri ohjelmointikielillä. Muutamia esimerkkejä suosituista sisällönhallinta- ja julkaisujärjestelmistä ja niiden hyödyntämistä ohjelmointikielistä ovat esimerkiksi Drupal(php), WordPress(php), Umbraco(.Net C#), Sharepoint(.Net), Liferay(Java) ja Alfresco(Java). Valitun teknologian tulee siis olla integroitavissa myös muillekin kuin .Net-teknologian kielille.

Microsoft Sync Framework

Sync Framework on Microsoftin kehittämä alusta tiedonsiirtoon ja synkronointiin. Alustaa mainostetaan erittäin joustavana mahdollistaen kaikkien eri tietotyyppien, synkronoinnin kaikkien sovellusten välillä käyttäen mitä tahansa protokollaa ja mitä tahansa verkkoyhteyttä. Alustan perustana ovat niin sanotut synkronoinnintarjoajat (provider). Tarjoajat mahdollistavat erityyppisten datalähteiden, kuten esimerkiksi tietokoneen tiedostojen, osallistumisen synkronointiin. Alustan mukana toimitetaan valmiit tarjoajat tietokantojen, tiedostojen sekä atom- ja rss-verkkosyötteiden synkronointiin. Lisäksi sovelluskehittäjät voivat luoda tarjoajia lisää tarvittaessa. [22.]

Sync Frameworkissä tietojen synkronointiin osallistuvat järjestelmät jaetaan kolmeen eri osallistujatyyppiin riippuen järjestelmän ominaisuuksista. Osallistujatyyppi kuvaa sitä, millä tavoin järjestelmää varten voidaan toteuttaa synkronoinnintarjoaja. Täydeksi osallistujaksi (full participant) kutsutaan järjestelmää, johon on mahdollista kehittää tarjoajasovellus ja joka mahdollistaa datan tallentamisen ja lukemisen. Esimerkkejä tästä ovat tietokone ja älypuhelin. Osittainen osallistuja (partial participant) tarjoaa mahdollisuuden datan tallentamiseen ja lukemiseen, mutta siihen ei voida asentaa ohjelmia. Tyyppiesimerkki tällaisesta järjestelmästä on digitaalikamera. Kolmas tyyppi on yksinkertainen osallistuja (simple participant). Tällaisesta järjestelmästä voidaan ainoastaan lukea dataa, mutta siihen ei voida tallentaa dataa eikä siihen voida asentaa ohjelmia. Esimerkkejä tällaisista järjestelmistä ovat erilaiset WWW-syötteet ja WWW-sovelluspalvelut, jotka ainoastaan palauttavat tietoa. [22.]

Tiedonsiirtomoduulin tapauksessa synkronoinnin osapuolet olisivat sisällönhallintajärjestelmä ja pilvipalvelualustalle toteutettu sovellus. Microsoft Sync Frameworkin hyödyntäminen moduulin osana vaatisi tarjoajien ohjelmoimista sekä pilvijulkaisualustalle että lähdejärjestelmään. Pilvipalvelualustalle tehty sovellus täyttää täyden osallistujan kriteerit tarjoamalla mahdollisuuden synkronointitarjoajan ohjelmoimiseen sekä datan tallentamiseen. Useimmissa tapauksissa myös sisällönhallintajärjestelmät mahdollistavat sekä datan tallentamisen että synkronointitarjoajan toteuttamisen.

Sync Framework suorittaa tietojen synkronoinnin metatietojen avulla. Alusta tallentaa tietoihin liittyvää metatietoa automaattisesti. Tätä metatietoa verrataan synkronointitilanteessa muiden osapuolien metatietoihin kyseisestä tiedosta. Näin voidaan päätellä, missä on tiedon uusin versio ja keiden synkronointiin kuuluvien osapuolten kuuluisi päivittää omat tietonsa. Erilaisia kerättyjä metatietoja ovat muun muassa päivityksen yhteydessä kasvatettava versionumero ja jokaiselle tiedolle yksilöllinen tunniste. [22.]

Tiedonsiirtomoduulin kannalta Sync Frameworkin suurin ongelma on se, että se on puhdas Microsoft-ratkaisu ja sen integroiminen useisiin eri teknologioilla toteutettuihin sisällönhallintajärjestelmiin vaikutti työläältä ja hankalalta. Alustan käytöstä muiden kuin Microsoft-teknologioiden kanssa ei löydy juurikaan esimerkkejä. Sync Frameworkin WWW-dokumentaatio on myös melko heikko. Alusta suorittaa myös suuren osan synkronointiin liittyvistä asioista automaattisesti, ja sen toiminnan ymmärtäminen ja siihen vaikuttaminen saattaisi muodostua liian työläiksi.

SQL Azure Data Sync

SQL Azure Data Sync on aiemmin esitellyn Microsoft Sync Frameworkin päälle rakennettu palvelu. Se mahdollistaa SQL Server relaatiotietokantojen synkronoimisen pilvipalvelualustalla sijaitsevaan Azure SQL -relaatiotietokantaan. Palvelu voidaan ottaa käyttöön asentamalla synkronoitavalle ei-pilvipohjaisille SQL Server -tietokantapalvelimille agenttiohjelmisto, joka mahdollistaa yhteydenpidon pilvitietokantaan. Tämän jälkeen Windows Azure Platform -tilin hallintasivustolla luodaan niin sanottu agenttiavain. Agenttiavain toimii tunnisteena, jonka avulla ei-pilvipohjainen tietokantapalvelin osaa ottaa yhteyden Azure SQL -tietokantapalveluun. Luotu agenttiavain asetetaan ei-pilvipohjaisen tietokantapalvelimen agenttiohjelmistoon. Seuraavaksi

agenttiohjelmistolla valitaan yksittäiset tietokannat, joita voidaan synkronoida. Näiden vaiheiden jälkeen pilvipalvelualustan hallintasivustolla voidaan rekisteröidä ei-pilvipohjaiset tietokannat synkronoitaviksi. Lopuksi hallintasivustolla muodostetaan halutuista tietokannoista synkronointiryhmä. Synkronointiryhmä sisältää pilvessä ja sen ulkopuolella sijaitsevat relaatiotietokannat, ja se synkronoi tietokannat halutun väliajan välein. [23.]

SQL Azure Data Sync on puhtaasti relaatiotietokantojen synkronointiin kehitetty ratkaisu. Tämä on myös sen pahin rajoite. Lisäksi SQL Azure pystyy ainoastaan synkronoimaan dataa Microsoftin SQL Server -ja Azure SQL -tietokantojen välillä.

Valittu toteutusmenetelmä

Näiden vaihtoehtojen pohjalta päädyttiin siihen, että koko tiedonsiirtomoduuli suunnitellaan ja toteutetaan itse. Valintaan vaikuttavana tärkeimpänä perusteena pidettiin sitä, että alusta loppuun asti tarkoitusta varten suunniteltu ja toteutettu ohjelmisto on luultavimmin yksinkertaisin jatkokehittää. Lisäksi se palvelee tarkoitustaan mahdollisimman tehokkaasti ilman turhia ominaisuuksia. Tutkimuksissa Sync Framework osoittautui liian Microsoft-sidonnaiseksi lähdejärjestelmäintegraatiota ajatellen. Integraatio eri ohjelmointikielillä toteutettuihin sisällönhallintajärjestelmiin olisi saattanut olla liian haastavaa. Lisäksi Sync Framework on suunniteltu pääasiassa eri laitteiden välisten tietojen synkronointiin ja se sisältää turhan paljon ominaisuuksia melko yksinkertaista tiedonjulkaisuprosessia ajatellen. SQL Azure Data Sync rajattiin soveltumattomana pois hyvin nopeasti. Syynä tähän oli se, että pilvijulkaisualusta ei tule käyttämään relaatiotietokantaa ollenkaan. Näin tiedostojen siirtämistä varten jouduttaisiin käyttämään turhaa tietokantaa, jonka ainoa tehtävä olisi toimia välivarastona siirretyille tiedoille. SQL Azure Data Sync on myös liian Microsoft-sidonnainen ratkaisu lähdejärjestelmäintegraatiota ajatellen.

4.3 WWW-sovelluspalvelu

Julkisen tiedonsiirtorajapinnan toteuttamiseen pilvipalvelualustalle päätettiin käyttää WWW-sovelluspalvelua eli web serviceä. WWW-sovelluspalvelulla tarkoitetaan ohjel-

mistojärjestelmää, jota voidaan käyttää yleisten web-protokollien ja teknologioiden avulla [24, s. 11]. Se on erittäin sopiva ratkaisu, koska julkaisualustalle on tarkoitus siirtää tietoa nimenomaan internetin välityksellä. Sen etuna on myös se, että sitä voidaan käyttää yleisten web-protokollien ja teknologioiden ansiosta lähes mistä tahansa järjestelmästä, jolla on yhteys internetiin. Useimmille ohjelmointikielille on myös saatavissa valmiit kirjastot WWW-sovelluspalveluiden hyödyntämiseen. Näin lähdejärjestelmäintegraatiot voidaan toteuttaa helposti eri ympäristöihin. Käytännössä WWW-sovelluspalvelu esiintyy sitä hyödyntävälle ohjelmistolle palveluna, joka tarjoaa tiettyjä operaatioita. Nämä operaatiot vastaavat ohjelmiston metodeita tai funktioita, joita voidaan käyttää internetin välityksellä. Useimmiten operaatiot ottavat syötteinään jonkinlaisen viestin ja myös voivat palauttaa sellaisen. Käytännön esimerkkinä WWW-sovelluspalvelusta voisi olla laskentapalvelu, jonka summaa-operaatio ottaa vastaan viestin, joka sisältää kaksi lukua. Tämän jälkeen palvelu summaa luvut ja palauttaa tuloksen vastausviestissä.

Tiedonsiirtomoduulin WWW-sovelluspalvelu osan suunnittelussa määriteltiin aluksi tarpeelliset operaatiot, joiden avulla tietoa voitaisiin siirtää pilvijulkaisualustalle. Näitä operaatioita voitaisiin kutsua internetin välityksellä ja näin lähdejärjestelmä voisi siirtää tietonsa julkaisualustalle. Tiedonsiirtomoduulin suunniteltiin operaatiot dataerän lähettämiseen ja datan lopulliseen julkaisuun. Dataerän lähettämisellä tarkoitetaan tilannetta, jossa lähdejärjestelmä haluaa siirtää joukon tiedostoja julkaisualustalle. Riippuen tiedostojen määrästä ja koosta voi lähdejärjestelmä pilkkoa alkuperäisen tiedostojoukon useaan dataerään. Lisäksi lähdejärjestelmän tulee voida pilkkoa yksittäinen iso tiedosto usean eri dataerän kesken. Lopulliselle julkaisulle haluttiin erillinen operaatio, koska dataa voidaan lähettää useassa vaihtelevan kokoisessa erässä ja yksi tiedosto voidaan joutua jakamaan useaan erään.

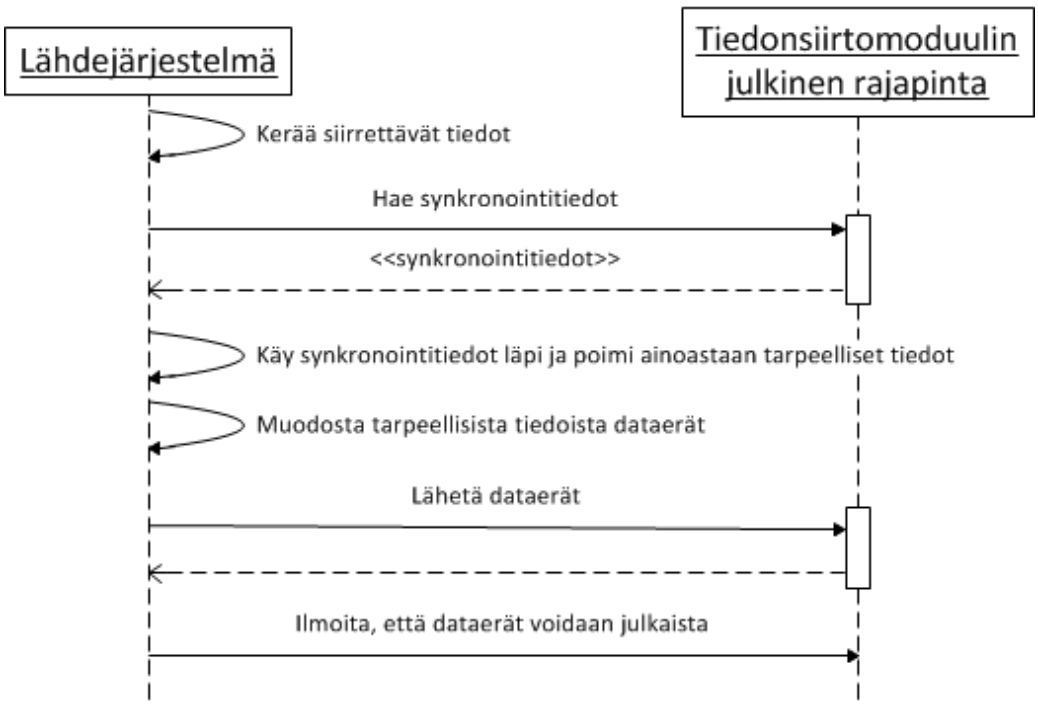
Pilvipalvelualustoille tehtäviä sovelluksia suunniteltaessa tulee ottaa huomioon alustan hinnoittelu. Melko pienet muutokset sovelluksessa saattavat aiheuttaa isoja eroja sovelluksen käytöstä veloitettavassa hinnassa. Yksi hinnoitteluun liittyvä huomio tiedonsiirtomoduulia suunniteltaessa oli siirretyn datan määrä. Windows Azure Platform -alustan laskutusperiaatteena ovat muun muassa alustalle siirretty datan määrä sekä sinne tallennetun datan määrä. Siirrettävän datan määrä haluttiin minimoida tarjoamalla lähdejärjestelmälle mahdollisuus kysyä tiedonsiirtomoduulilta, mitä tiedostoja julkaisualusta

tarvitsee. Tämä myös yksinkertaistaa lähdejärjestelmäintegraatiota. Lähdejärjestelmä voi lähettää kuvauksen kaikista tiedostoistaan tiedonsiirtomoduulille ja vastauksena se saa tiedon siitä, mitä kyseisistä tiedoista tarvitaan. Tätä toiminnallisuutta varten tiedonsiirtomoduuliin määriteltiin vielä operaatio synkronointitietojen pyytämistä varten.

Tiedonsiirtomoduulin tarjoamat julkiset operaatiot nimineen ovat siis:

- synkronointitiedon tarjoaminen (GetRemoteVersions)
- dataerän vastaanotto (UploadDataBatch)
- tiedostojen julkaisu (ProcessPublish).

Lähdejärjestelmän toiminta tyypillisessä julkaisutilanteessa olisi siis seuraavanlainen. 1) Kerätään tiedot, jotka halutaan lähettää pilvijulkaisualustalle. 2) Pyydetään tiedonsiirtomoduulilta synkronointitiedot niistä tiedoista. 3) Käydään saadut synkronointitiedot läpi ja päätellään siitä, mitä tiedostoja pilvijulkaisualustalle pitää oikeasti lähettää. 4) Muodostetaan tarvittavista tiedoista dataeriä. 5) Lähetetään dataerät tiedonsiirtomoduulille. 6) Ilmoitetaan tiedonsiirtomoduulille, että lähetetyt tiedostot voidaan julkaista. Kuvio 10 havainnollistaa tätä julkaisuprosessia.

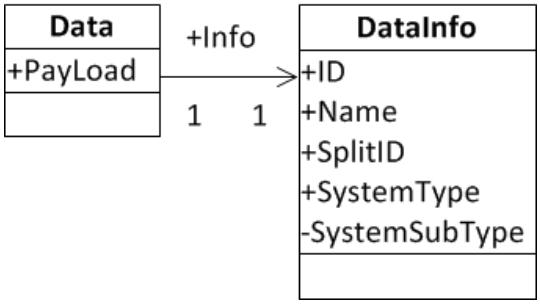


Kuvio 10. Julkaisuprosessin hahmotelma.

4.4 Palvelun viestit

Operaatioiden määrittelyn jälkeen siirryttiin suunnitelmaa tarkentamaan mallintamalla operaatioissa liikkuvat viestit. Viestien tulee kuljettaa tiedot, joita tarvitaan operaatioiden suorittamiseksi. Viestit ovat WWW-sovelluspalvelun kannalta tietyn muotoisia dokumentteja, joiden tehtävänä on siirtää tietoa palveluun ja sieltä pois. Monet toteutukset WWW-sovelluspalveluista käyttävät dokumenttien muotona xml-kuvauskieltä. Useimmiten moderneja ohjelmistokehitystyökaluja ja ohjelmointikehyksiä käytettäessä WWW-sovelluspalvelun viestit mallinnetaan olio-ohjelmoinnin luokkina. Näiden luokkien perusteella ohjelmistokehykset muodostavat lähes automaattisesti internetin välityksellä siirrettävät dokumentit. Sovelluskehittäjä käsittelee siis WWW-sovelluspalvelua kehittäessään ja käyttäessään lähes poikkeuksetta olioita, jotka toimivat viesteinä eri järjestelmien välillä.

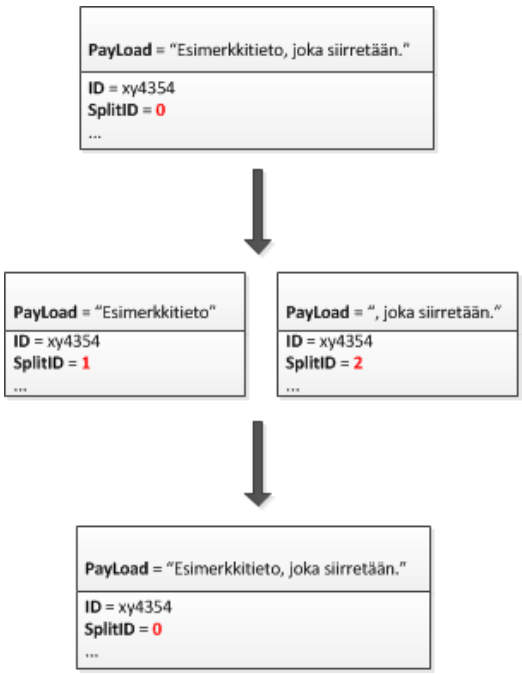
Aluksi suunniteltiin viesti, joka mallintaa yhden palveluun lähetetyn tiedon eli yhden data-erän osan. Tämän viestin nimeksi annettiin Data. Data-viestin tulee sisältää siirrettävä data tavumuodossa sekä kuvaus, jonka perusteella data voidaan tunnistaa. Tämän pohjalta Data-viestiin päätettiin sisällyttää erillinen kuvausviesti DataInfo. DataInfo-viesti sisältää tarvittavaa tietoa, jotta data voidaan tunnistaa ja sitä voidaan hyödyntää pilvijulkaisualustalla. DataInfo-viestiä on myös tarkoitus hyödyntää synkronointitietoja välitettäessä. Synkronointitietojen ei tarvitse sisältää itse dataa vaan riittää, että ne sisältävät riittävän kuvauksen siirrettävästä datasta. Kuvio 11 havainnollistaa Data- ja DataInfo viestejä uml-luokkadiagrammin muodossa.



Kuvio 11. Data- ja DataInfo-viestit uml-luokkadiagrammina.

DataInfo-viestin ominaisuuksiksi otettiin ID, joka on lähdejärjestelmän määrittämä uniikki tunniste datalle. Name-ominaisuus mahdollistaa selkokielisen nimen tarjoamisen. SystemType-ominaisuudella kuvataan, minkä tyyppinen lähdejärjestelmä on. SystemSubtype-ominaisuutta käytetään tunnistamaan tiedon tyyppi lähdejärjestelmän sisällä. SystemTypen ja SystemSubtypen avulla voidaan viesteihin kohdentaa lähdejärjestelmään ja sen tietotyyppien perusteella käsittelytoimenpiteitä. Esimerkiksi lähdejärjestelmän A lähettämille jpg-muotoisille kuville saatetaan haluta tehdä eri toimenpiteitä kuin saman järjestelmän png-kuville. Vastaavasti lähdejärjestelmällä B saattaa olla täysin eri käsittelytoimenpiteet samoille tietotyypeille.

SplitID on numero, jonka lähdejärjestelmä voi asettaa, jos se päättää pilkkoa yhden tiedon useaan eri osaan. SplitID:tä hyödyntämällä tiedonsiirtomoduuli osaa koostaa pilkotun tiedon takaisin alkuperäiseen muotoonsa. Kun yksittäinen tieto halutaan pilkkoa, jaetaan se paloihin ja sijoitetaan useaan Data-viestiin. Viestin DataInfo-ominaisuudessa ilmoitetaan SplitID:n avulla, minkä osan alkuperäisestä tiedosta kyseinen Data-viesti sisältää. Tieto on mahdollista koostaa takaisin alkuperäisen muotoonsa yhdistämällä saman ID:n omaavat tiedot SplitID:n mukaisessa järjestyksessä. Kuvio 12 havainnollistaa tiedon pilkkomista.

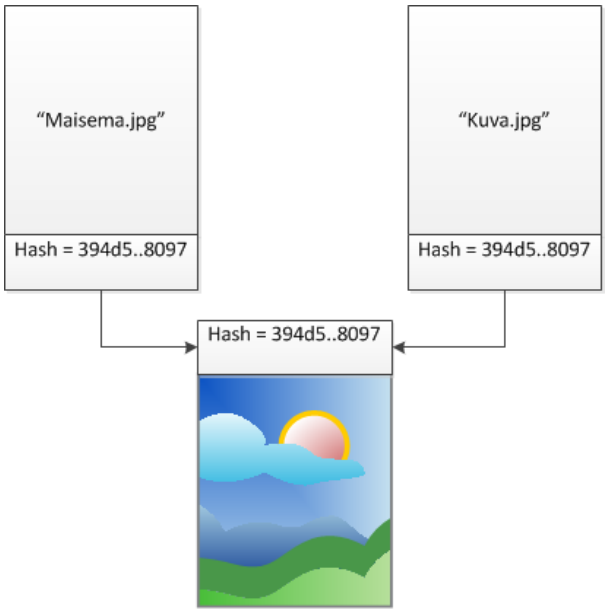


Kuvio 12. Esimerkki tiedon pilkkomisesta ja kokoamisesta SplitID-ominaisuuden avulla.

Kuviossa aloitetaan ylhäällä tilanteesta, jossa yksi tieto halutaan pilkkoa useampaan osaan. Tieto pilkotaan kahteen osaan ja osille asetetaan SplitID-ominaisuudet, jotka kertovat kuinka mones osa on kyseessä. Lopuksi pilkotut tiedot kootaan takaisin alkuperäiseen muotoon yhdistämällä dataosat SplitID tietojen mukaisessa järjestyksessä.

DataInfo-viestin SHA512ContentHash-ominaisuus sisältää lähdejärjestelmän laskeman SHA512-muotoisen kryptografisen tiivistefunktion DataInfoon liittyvästä tietosisällöstä. Kryptografinen tiivistefunktio on funktio, joka muodostaa sille syötetystä tiedostosta uniikin, niin sanotun hash-tunnisteen. Usein kryptografisia tiivistefunktioita käytetään tunnistamaan muutoksia tiedoissa. Tällöin tiedosta lasketaan aluksi hash-tunniste. Tämän jälkeen muutokset tiedossa voidaan havaita laskemalla uudestaan hash-tunniste tiedosta. Jos uusi ja vanha tunniste eroavat, on kyseistä tietoa muutettu.

Tiedonsiirtomoduulin ja pilvijulkaisualustan tapauksessa tiedosta laskettavaa hash-tunnistetta päätettiin käyttää minimoimaan järjestelmään tallennetun tiedon määrä. Tähän päädyttiin, koska pilvipalvelualustan yhtenä hinnoitteluperusteena on se, kuinka paljon sinne on tietoa tallennettu. On hyvin tyypillistä, että samaa tietoa käytetään useilla eri nimillä samassa järjestelmässä ja usein samasta tiedosta on useita kopioita. Esimerkkinä tästä on vaikkapa kuvatiedosto, joka saatetaan tallentaa eri nimillä useaan kertaan. Hash-tunnistetta voidaan käyttää tallennetun tietomäärän minimoimiseen. Koska hash-tunniste on tiedolle yksilöllinen, voidaan uniikin tiedon tunnistamiseen käyttää hash-tunnistetta. Jokainen uniikin hash-tunnisteen omaava tieto tallennetaan kerran, ja tämän jälkeen tietoon voidaan viitata useilla eri nimillä. Kuvio 13 havainnollistaa tätä periaatetta, jossa sama kuva on tallennettuna kahdella eri nimellä.



Kuvio 13. Samaan tietoon viittaaminen hash-tunnisteen perusteella.

Erilaiset kryptografiset tiivistefunktiot muodostavat eripituisia hash-tunnisteita. Tässä työssä käytetty SHA512 muodostaa 512-bittisen tunnisteen. Koska tunnisteen pituus on rajattu, on olemassa teoreettinen mahdollisuus, että eri tiedoille voidaan laskea sama hash-tunniste. Tämä johtuu siitä, että äärettömän suuri syötejoukko kuvataan rajallisella tunnisteella. Tilannetta, jossa kahdella eri syötteellä saadaan sama tunniste, kutsutaan törmäykseksi (collision).

Kirjallisuudessa on raportoitu vastaavasta tavasta tunnistaa uniikit tiedostot MD5-tiivistefunktion avulla [25, s. 583]. Kyseisessä toteutuksessa MD5-funktion 128-bittinen hash-tunniste ei ollut pituudeltaan riittävä ja muutamat eri tiedot tuottivat saman hash-tunnisteen [25, s. 583]. Tilanne, jossa törmäyksen todennäköisyys kasvaa yli 50 %:n, voidaan arvioida kaavalla.

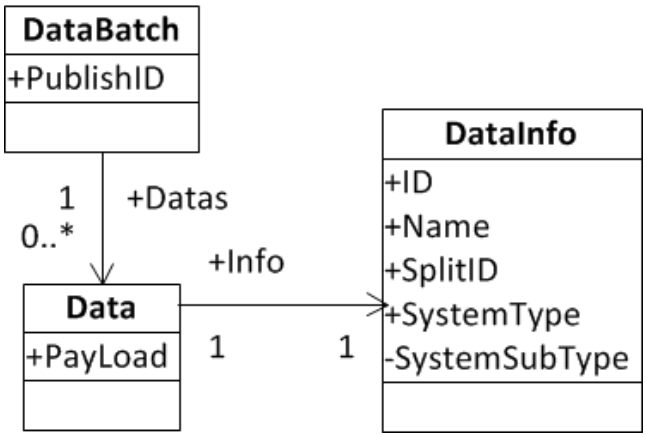
$$N = 2^{\frac{L}{2}}$$

N on eri tietojen lukumäärä

L on hash-tunnisteen bittimäärä. [26, s. 181–182.]

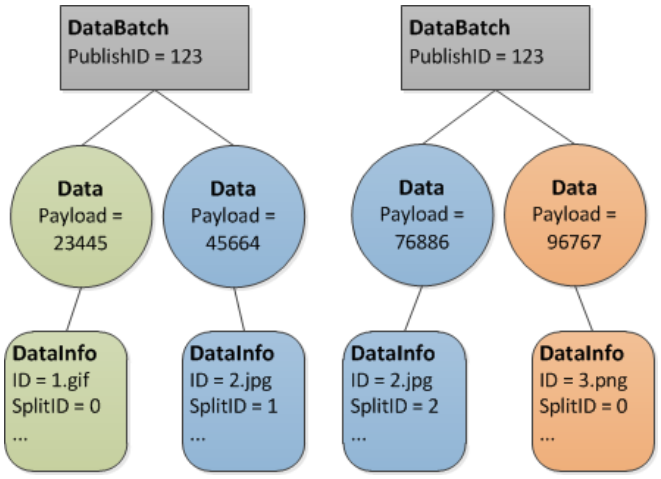
Tähän työhön valittiin SH512, koska se tarjoaa erittäin pienen todennäköisyyden törmäyksille. Kaavan perusteella 512-bittisellä hash-tunnisteella voidaan ilmaista 10^{77} eri tietoa, ennen kuin törmäys tulee todennäköiseksi.

Koska dataa haluttiin lähettää erissä, suunniteltiin viesti, joka sisältää useita Data-viestejä. Tämän viestin nimeksi annettiin DataBatch, ja sitä käytetään dataerän lähetysoperaatiossa (UploadDataBatch). DataBatch-viestiin sisällytettiin myös julkaisutunnus PublishID, jonka perusteella useat dataerä ovat yhteenkuuluvia ja osa samaa julkaisua. Kuvio 14 havainnollistaa DataBatch-viestiä.



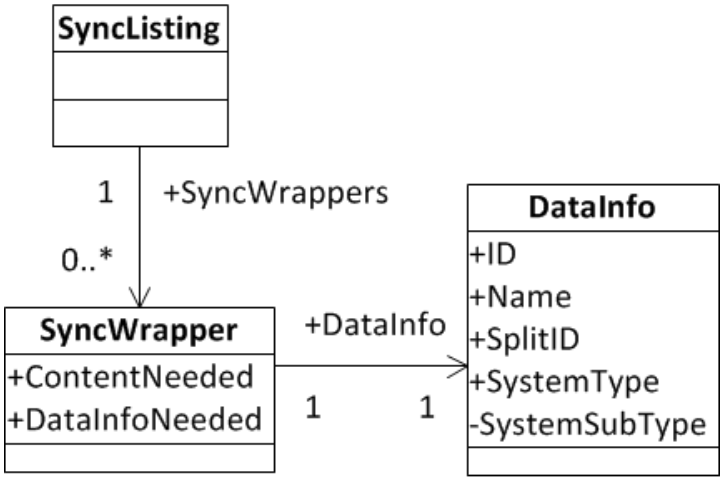
Kuvio 14. DataBatch viesti uml-luokkadiagrammina.

Kuviossa 15 on kuvattuna tilanne, jossa on kaksi dataerää. Nämä dataerät sisältävät kolme eri tiedostoa: 1.gif, 2.jpg ja 3.png. Tiedostoista 2.jpg on pilkottu kahteen eri osaan, jotka lähetetään eri dataerissä. Dataerien PublishID on sama, jotta julkaistavat tiedot voidaan yhdistää toisiinsa.



Kuvio 15. Dataerien käyttö.

Synkronointitietojen pyytäminen tiedonsiirtomoduulilta suoritetaan GetRemoteVersions-operaatiolla. Operaation viestinä toimii molempiin suuntiin SyncListing. SyncListingin tehtävänä on välittää kuvaukset mahdollisesti siirrettävistä tiedoista ja kertoa mitä niistä järjestelmä tarvitsee tiedon julkaisemiseksi. SyncListing sisältää kokoelman SyncWrapper-synkronointikääreitä, joista jokainen sisältää yhden DataInfo-viestin. Kuvio 16 havainnollistaa SyncListing- ja SyncWrapper-viestejä.



Kuvio 16. SyncListing- ja SyncWrapper-viestit uml-luokkadiagrammina.

SyncWrapper kertoo, mitä siihen linkitetyn DataInfo-viestin kuvaaman datan julkaisemiseen tarvitaan. SyncWrapperin ominaisuus ContentNeeded kertoo, että kyseisen

tiedoston julkaisemiseen tarvitaan itse data. DataInfoNeedeed taas ilmaisee, että datasta tarvitaan sitä kuvaavat tiedot.

Synkronointilistausta pyydettäessä tiedonsiirtomoduuli vertaa DataInfo-viestin SHA512ContentHash- sekä ID-ominaisuutta pilvijulkaisualustalla jo oleviin tietoihin. Jos järjestelmästä löytyy kyseinen hash-tunniste viittauksella samaan ID-tunnisteeseen, on kyseessä tilanne, jossa samaa tietosisältöä yritetään julkaista samalla tunnisteella useaan kertaan. Tällöin järjestelmä ilmoittaa SyncWrapper-viestin ContentNeeded- ja DataInfoNeeded-ominaisuuksien avulla, että lähdejärjestelmän ei tarvitse lähettää kyseistä tietoa. Mikäli kyseinen hash-tunniste löytyy järjestelmästä mutta siihen viitataan eri ID:llä, tarvitsee tiedonsiirtomoduulin ainoastaan linkittää uusi ID jo olemassa olevaan tietoon kuvion 13 mukaisesti. Taulukko 4 esittää mahdolliset hash-tunniste ja ID yhdistelmät ja niitä vastaavat arvot ContentNeeded ja DataInfoNeeded ominaisuuksille.

Taulukko 4. Hash-tunniste ja ID parien vaikutukset ContentDeeded ja DataInfoNeeded ominaisuuksiin.

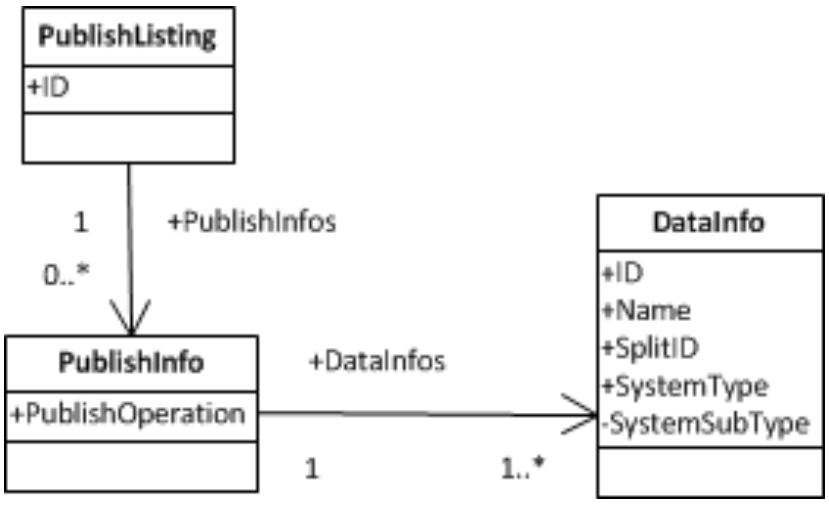
Hash pilvijulkaisualustalla	ID pilvijulkaisualustalla	ContentNeeded	DataInfoNeeded
Kyllä	Kyllä	False	False
Kyllä	Ei	False	True
Ei	Ei	True	True
Ei	Kyllä	True	True

Sen jälkeen kun tiedonsiirtomoduuli on täydentänyt ContentNeeded ja DataInfoNeeded ominaisuudet SyncWrapperille, se palauttaa täydentämänsä synkronointilistauksen lähdejärjestelmälle. Nyt lähdejärjestelmä voi hyödyntää ContentNeeded ja DataInfoNeeded ominaisuuksia päättäessään mitä tietoja se lähettää. Liitteessä 2 havainnollistetaan tätä periaatetta. Liitteessä lähdejärjestelmä pyytää synkronointitietoja kahdesta tiedostosta 7.jpg ja 8.jpg. Tiedonsiirtomoduuli tarkastaa, löytyykö siltä jo kyseisiä tietoja ja viittauksia. Tiedoston 7.jpg tapauksessa itse tieto löytyy, mutta eri viittauksella. Tiedosto 8.jpg löytyy kokonaisuudessaan. Tiedonsiirtomoduuli merkitsee paluuviestiin, että 7.jpg julkaisemiseksi tarvitaan ainoastaan DataInfo ja 8.jpg julkaisemiseksi ei tarvita mitään.

Tarpeellisten tietojen lähetyksen jälkeen lähdejärjestelmän tulee ilmaista, että sen lähettämät tiedot voidaan julkaista. Tätä varten suunniteltiin julkaisulistausviesti Publish-

Listing. Lähdejärjestelmän tehtävänä on koostaa PublishListing niiden tietojen perusteella, jotka se on julkaisualustalle lähettänyt ja jotka se haluaa julkaistavan. Tämän lisäksi PublishListing mahdollistaa pelkän DataInfo-viestin siirtämisen julkaisualustalle tapauksissa, joissa koko datasisältöä ei tarvinnut siirtää, koska se sijaitsi jo julkaisualustalla. Tällöin pelkän DataInfo-viestin perusteella voidaan luoda viittaus jo olemassa olevaan tietoon. PublishListing-viestin ID-ominaisuus on julkaisulle uniikki tunniste, ja se on myös sama kuin julkaistujen DataBatch-viestien PublishID.

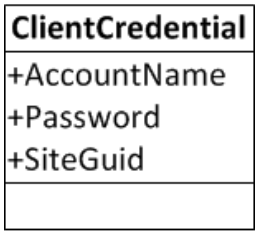
PublishListing sisältää tiedon julkaistavista tiedoissa useissa PublishInfo-viesteissä. PublishInfo-viestin tarkoituksena on kuvata, kuinka yksi kokonainen tieto on palveluun lähetetty. Tästä syystä PublishInfo voi sisältää viittauksen moneen DataInfo-viestiin. Näillä yhden PublishInfo-viestin DataInfoilla kuvataan sitä, kuinka tiedosto on lähetysvaiheessa pilkottu. Jos tiedostoa ei ole pilkottu, sisältää PublishInfo viittauksen ainoastaan yhdeen DataInfoon. Jos taas tieto on pilkottu, viitataan PublishInfossa pilkotun tiedon kaikkiin DataInfoihin. Tämän avulla pilvijulkaisualusta voi koostaa tiedon takaisin alkuperäiseen muotoonsa. PublishInfo kuvaa PublishOperation ominaisuuden avulla, minkälainen julkaisuoperaatio on kyseessä. Mahdollisia arvoja julkaisuoperaatiolle ovat uuden tiedon lisäys (Add), jo olemassa olevan tiedon viittauksen päivitys (UpdateReference) ja tiedon poisto (Remove). Kuvio 17 havainnollistaa PublishListing- ja PublishInfo-viestejä.



Kuvio 17. PublishListing ja PublishInfo viestit uml-luokkadiagrammina.

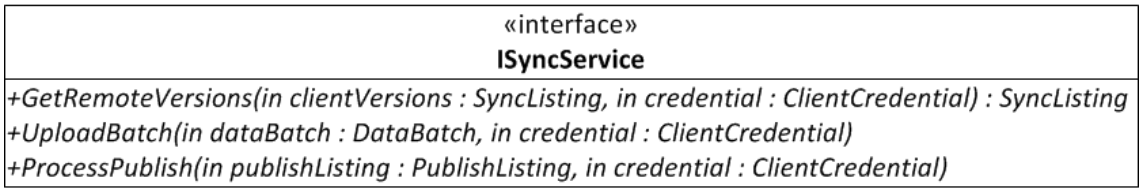
Liitteessä 3 esitellään tilanne, jossa pilvijulkaisualustalle lähetetään dataa kahdessa erässä ja julkaistaan ne. Tietojen julkaisua varten muodostetaan julkaisulistaus (PublishListing). Julkaisulistauksessa dataerien pilkotut osat ryhmitellään PublishInfo-ominaisuuden avulla. Lisäksi PublishInfo-ominaisuudella kuvataan, että kyseessä on tiedostojen lisäys.

Tiedonsiirtomoduulin täytyy pystyä tunnistamaan, kuka siihen lähettää dataa ja onko lähettäjällä tähän oikeus. Tämä tunnistaminen perustuu julkaisualustan käyttäjämalliin, joka on jaettu käyttäjiin ja käyttäjien sivustoihin. Näiden tietojen välittämiseksi suunniteltiin ClientCredential-viesti. ClientCredential-viesti välitetään jokaisen tiedonsiirtomoduulin operaation mukana, jotta operaation käyttäjä voidaan todentaa. ClientCredential pitää sisällään asiakkaan tunnisteen (AccountName), salasanan (Password) ja sivuston tunnisteen (SiteGuid). Kuvio 18 esittää ClientCredential-viestin rakenteen.



Kuvio 18. ClientCredential-viesti uml-luokkadiagrammina.

Viestien suunnittelun jälkeen voitiin tarkentaa tiedonsiirtomoduulin julkinen rajapinta lopulliseen muotoonsa. Kuvio 19 esittää rajapinnan uml-luokkadiagrammin avulla. Liitteessä 1 on kuvattuna kaikki tiedonsiirtomoduulin käyttämät viestit ja niiden suhteet toisiinsa.



Kuvio 19. Tiedonsiirto-moduulin julkinen rajapinta uml-luokkadiagrammina.

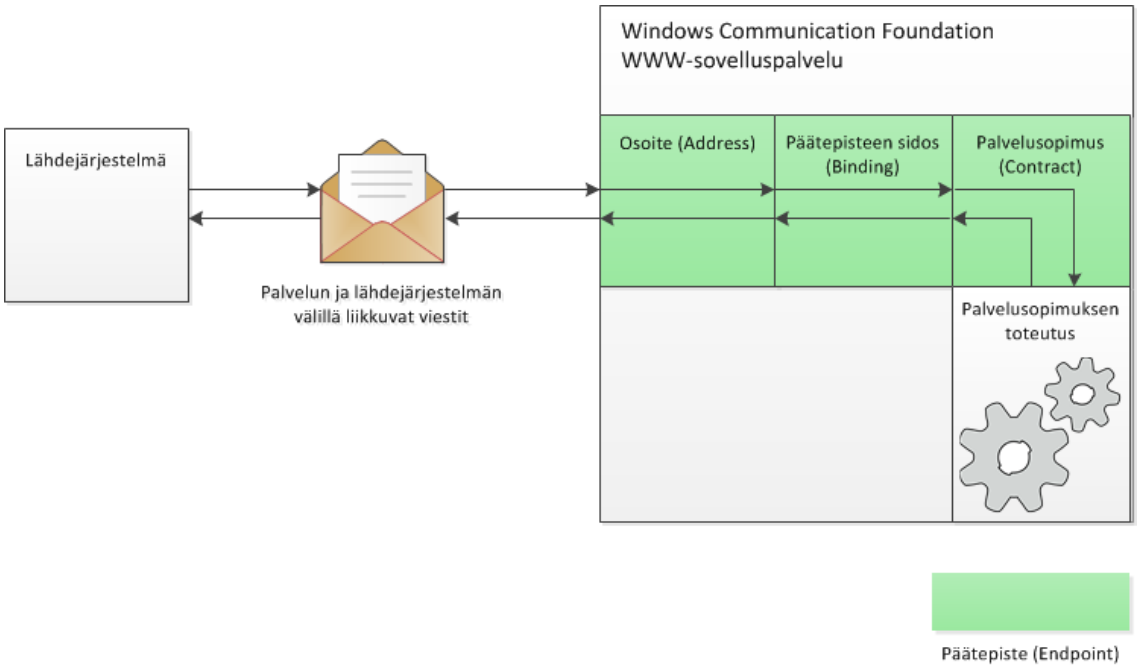
5 Toteutus ja testaus

5.1 Toteutus

Tiedonsiirtomoduulin julkinen rajapinta toteutettiin osana pilvijulkaisualustaa Windows Azure Platform -ympäristöön. Alustalle on helpointa kehittää sovelluksia käyttämällä Visual Studio -ohjelmistokehitysympäristöä. Lisäksi tarvitaan Windows Azure tools for Visual Studio -lisäosa, jonka avulla sovelluksia voidaan testata paikallisessa kehitysympäristössä ennen kuin ne siirretään oikeaan pilviympäristöön. Sovellusten paikallinen testaaminen on toteutettu Windows Azure -emulaattorin avulla. Emulaattori luo koneelle melko pitkälle oikeaa Windows Azurea vastaavan ympäristön. On kuitenkin huomioitava, että emulaattori ja oikea ympäristö eroavat toisistaan tietyissä tilanteissa. Ainakaan toistaiseksi ei voida pitää itsestään selvänä, että emulaattorissa toiminut sovellus toimisi automaattisesti myös oikeassa pilviympäristössä. Tästä syystä on tärkeää testata kehitettävää sovellusta jatkuvasti myös oikeassa Windows Azure -ympäristössä. Tulevaisuudessa pilvijulkaisualustan testaus tullaan suorittamaan automaattisesti pilviympäristössä.

WWW-sovelluspalveluiden kehittäminen Microsoftin .Net -ympäristössä on melko suoraviivaista. .Net Framework sisältää Windows Communication Foundation (WCF) -alustan, jonka avulla WWW-sovelluspalveluita voidaan toteuttaa. WCF:ää käytettäessä WWW-sovelluspalvelun viestit ja operaatiot mallinnetaan luokkien ja rajapintojen avulla. Operaatioita varten luodaan rajapinta (interface), johon merkityt metodit kuvaavat WWW-sovelluspalvelun operaatioita. Tätä rajapintaa kutsutaan palvelusopimukseksi (service contract). Rajapinnan operaatiot toteutetaan WWW-sovelluspalvelun perusrakenteen muodostavassa luokassa. Myös viestien mallinnus tehdään luokkien avulla.

Kun WWW-sovelluspalvelun rakenne on toteutettu, tulee WCF:lle kertoa, millä tavoin palvelua voidaan käyttää ja missä se sijaitsee. WCF-alustalla tätä kutsutaan päätepisteen (endpoint) määrittelyksi. Päätepiste määrittelee WWW-sovelluspalvelun osoitteen (address), palvelusopimuksen (contract) ja sidoksen (binding). Kuvio 20 havainnollistaa WCF:lla toteutetun WWW-sovelluspalvelun rakennetta ja sitä, kuinka päätepiste muodostuu.



Kuvio 20. Windows Communication Foundation WWW-sovelluspalvelun rakenne.

Osoite kertoo missä url-osoittessa palvelu sijaitsee. Palvelusopimuksella tarkoitetaan rajapintaa, jossa WWW-sovelluspalvelun operaatiot on määritelty. Päätepisteen sidos eli binding määrittelee, millä tavoin palvelua voidaan käyttää. Sidos määrittelee muun muassa mitä tiedonsiirtoprotokollaa, enkoodausta ja tietoturvaominaisuuksia WWW-sovelluspalvelu käyttää. WCF sisältää useita valmiita binding-vaihtoehtoja eri tilanteisiin. [27, s. 77–80.] Tiedonsiirtomoduulin tapauksessa käytettäväksi sidokseksi valittiin wsHttpBinding. Valintaa perusteltiin sillä, että wsHttpBinding on yhteensopiva muidenkin kuin Microsoft-teknologioiden kanssa [27, s. 79]. Lisäksi se tarjoaa kattavat tietoturvaominaisuudet mahdollistaen esimerkiksi salatun https-protokollan käytön [27, s. 79–80]. Tulevaisuudessa on myös mahdollista käyttää tiedonsiirtomoduulissa useita eri sidos tyyppejä. Näin on mahdollista käyttää esimerkiksi Microsoft-ympäristöille optimoitua netTcpBindingia. [28, s. 80.] WCF-pohjaisen WWW-sovelluspalvelun päätepisteen määrittely on melko työlästä ja se voidaan suorittaa joko konfiguraatiotiedoston avulla tai ohjelmallisesti. Näistä tavoista konfiguraatiotiedoston käyttö on suositeltavampaa, sillä se mahdollistaa päätepisteen ominaisuuksien muuttamisen dynaamisesti ilman, että ohjelmakoodiin tarvitsee koskea [27, s. 82]. Tästä syystä tiedonsiirtomoduulin päätepisteen konfigurointi päädyttiin toteuttamaan konfiguraatiotiedoston avulla.

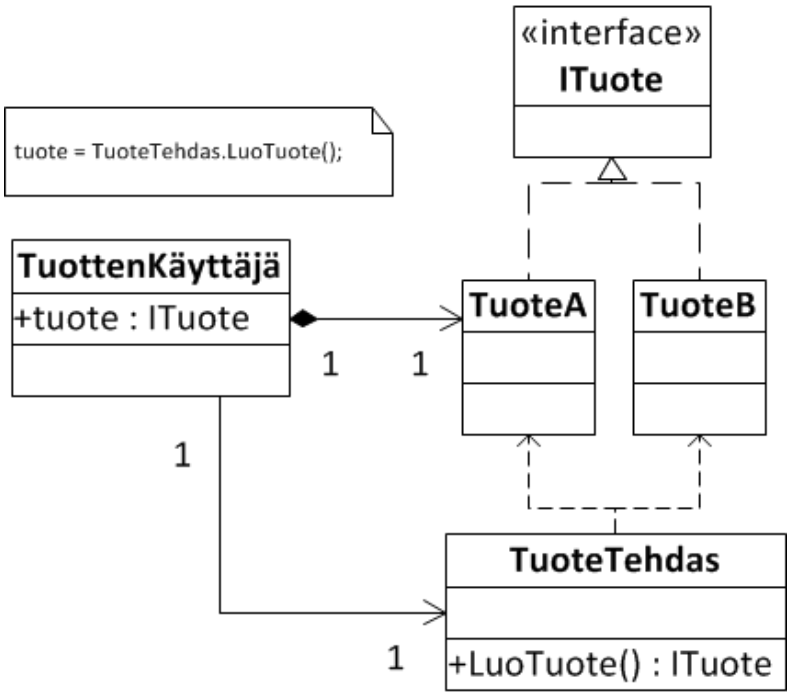
Tiedonsiirtomoduulin päätepisteen konfiguroinnissa jouduttiin ottamaan huomioon palvelun tietoturva ja mahdollisesti suuret tietomäärät. Koska yhteys tiedonsiirtomoduuliin haluttiin pitää salattuna, asetettiin wsHttpBinding käyttämään siirtotason salausta konfiguraation avulla. Tämä tarkoittaa sitä, että kaikki liikenne palvelun ja sen käyttäjien välillä kulkee ssl-salatun https-protokollan avulla. WCF-päätepiste voi asettaa useita rajoituksia sinne lähetetyn tiedon määrälle ja viestien monimutkaisuudelle. Jotta palvelu voisi vastaanottaa mahdollisesti suurikokoisia Data-viestejä, konfiguroitiin päätepisteen maksimiviestikoko kahteen gigatavuun. Myös viestien taulu-ominaisuuksien pituus asetettiin kahteen miljardiin elementtiin, jotta Data-viestin tavutaulu muotoinen tietosisältö saataisiin siirrettyä kokonaisuudessaan.

WCF-pohjainen WWW-sovelluspalvelu voidaan asentaa Windows Azureen kahdella eri tavalla. Se voidaan asentaa web- tai työrooliin. Sovellusta kehitettäessä päädyttiin aluksi asentamaan tiedonsiirtomoduuli työrooliin. Työroolin etuna on se, että siihen voidaan ottaa yhteyttä muillakin kuin http(s)-protokollilla. Tämän arveltiin olevan hyödyllinen ominaisuus, jos joskus haluttaisiin hyödyntää muitakin protokollia http:n lisäksi. Työroolia käytettäessä törmättiin kuitenkin ongelmiin siinä vaiheessa, kun sovellusta alettiin testaamaan oikeassa Azure-ympäristössä emulaattorin sijaan. Tiedonsiirtomoduuliin ei saanut millään yhteyttä. Ongelmaa koitettiin selvittää muun muassa Microsoftin Azure-dokumentaation avulla. Tässä yhteydessä tuli esille se, että alustan dokumentaatio on toistaiseksi melko suppea. Lähes poikkeuksetta kaikissa esimerkeissä näytettiin ainoastaan, kuinka työrooliin saadaan yhteys emulaattoriympäristössä. Lopulta ongelman pohjalta laadittiin tukipyyntö Microsoftille. Ongelmaan saatiin ratkaisuehdotus Microsoftilta melko pian.

Ratkaisuehdotuksesta selvisi, että kaikki työrooliin tulevat yhteydet tulee erikseen sallia sovelluksen käynnistyksen yhteydessä. Toistaiseksi tätä varten tulee luoda erillinen apusovellus joka suoritetaan työroolin käynnistyksen yhteydessä. Apusovelluksen tehtävänä on ohjelmallisesti varata oikeudet joita työrooli tarvitsee. [29] Tapa, jolla yhteyksiä työrooliin sallitaan, vaikuttaa hieman keskeneräiseltä ja hankalalta. Pienen apusovelluksen tekeminen ja ylläpitäminen ei vaikuttanut tässä vaiheessa järkevältä. Tästä syystä palvelu päätettiinkin asentaa WWW-rooliin. WWW-roolin etuna on se, että se käyttää IIS-sovelluspalvelinta ja WCF WWW -sovelluspalvelut useimmiten asennetaan IIS-sovelluspalvelimille. Näin palvelun hallinta on Windows Azure -ympäristössä hyvin

samankaltaista kuin perinteisessä konesaliratkaisussa. Sovelluksen siirtäminen WWW-roolista työrooliin on yksinkertaista ja tulevaisuudessa se voidaan tehdä jos tarvetta ilmenee. Sovellusta voidaan jopa helposti pitää sekä WWW-roolissa että työroolissa.

Tiedonsiirtomoduulin laajennettavuus erilaisia lähdejärjestelmiä varten toteutettiin niin sanotulla tehdas-suunnittelumallilla (factory). Suunnittelumallit ovat yleiskäyttöisiä ohjelmistorakenteita, joita hyödyntämällä sovelluksesta saadaan muun muassa joustava ja selkeärakenteinen [30, s. 1–2]. Factory-mallin avulla voidaan luoda tietyn rajapinnan toteuttavia olioita tietämättä tarkalleen, minkä tyyppisiä oliot ovat. Olioita luodaan niin sanonut tehtaan avulla. Kuvio 21 havainnollistaa factory-suunnittelumallia kuvitteellisen tuotetehtaan avulla.



Kuvio 21. Factory-suunnittelumalli kuvitteellisen tuotetehtaan avulla [30, s. 110].

Kuviossa 21 TuotteenKäyttäjä tarvitsee Ituote-rajapinnan toteuttavan olion. Se ei luo oliota itse vaan pyytää sitä TuoteTehtaalta. Näin tuotetehtaan tehtäväksi jää päättää, minkä tyyppisen ITuotteen se luo. Tässä esimerkissä se voi luoda joko TuoteA- tai TuoteB-tyypin ITuotteen.

Tiedonsiirtomoduuli hyödyntää factory-mallia päättäessään, miten se käsittelee lähdejärjestelmän julkaisemia tietoja. Tiedonsiirtomoduuli määrittelee tietojen käsittelyn läh-

dejärjestelmän (SystemType) ja sen alitietotyypin (SystemSubtype) perusteella. Tätä varten toteutettiin kaksitasoinen tehdas-malli, jossa ensin SystemTypen perusteella valitaan lähdejärjestelmäkohtainen tehdas, joka luo SystemSubtypen perusteella sopivan käsittelijän tiedolle. Liittessä 4 on esiteltynä moduulin tehdastoteutus. Kun tiedonsiirtomoduulin piiriin halutaan liittää uusi lähdejärjestelmä, luodaan tälle lähdejärjestelmälle uusi tehdas. Tämän tehtaan tehtävänä on tarjota tiedon käsittelijöitä kyseiselle lähdejärjestelmälle. Liitteessä 5 havainnollistetaan uml-sekvenssikaaviolla tilannetta, jossa tiedolle, jonka lähdejärjestelmä on A ja tietotyyppi B, haetaan käsittelijä.

5.2 Hyväksi havaitut käytännöt

Tiedon käsittely ja säilytys tiedonsiirtomoduulissa ja pilvijulkaisualustalla tapahtuu Windows Azuren binääriobjektien ja taulujen avulla. Pilvijulkaisualusta tallentaa tietosisällön blobina, ja siihen liittyvät viittaukset tallennetaan tauluun. Azuren tallennusjärjestelmiä käytettäessä tulee aina tiedostaa niiden käytöstä aiheutuvat kustannukset. Etenkin tauluja käytettäessä tulee olla tarkkana, koska taulut toimivat monesti relaatiotietokannan korvaajana ja niihin tehdään paljon kyselyitä. Tiedonsiirtomoduuli ja pilvijulkaisualusta käsittelevät tauluja ja binääriobjekteja StorageClient-kirjaston avulla. StorageClient-kirjasto on Microsoftin .Net-ympäristöä varten kehittämä työkalukokoelma, jonka avulla voidaan käyttää Azuren tallennusjärjestelmiä. Kirjasto tarjoaa kokoelman luokkia, joiden avulla tallennusjärjestelmiä käsitellään. Tiedonsiirtomoduulille kriittisiä kohtia ovat synkronointitietojen hakeminen ja tietojen lopullinen julkaisu. Näissä tilanteissa taulujärjestelmään syntyy helposti suuri määrä laskutettavia transaktioita eli kirjoitus- ja lukuoperaatioita. Niiden määrää voidaan kuitenkin minimoida muutamin keinoin.

Synkronointitietoja pyydettäessä käydään läpi kaikki jo olemassa olevat viittaukset. Käytännössä lähdejärjestelmän lähettämää listausta tiedoistaan verrataan pilvialustalla oleviin viittauksiin. Tämän tyyppisissä tilanteissa on tärkeää saada ensin ladattua taulujärjestelmästä kaikki tarpeelliset tiedot mahdollisimman pienellä transaktiomäärällä. Tietojen latauduttua voidaan kahta muistissa olevaa kokoelmaa verrata toisiinsa nopeasti. Azuren tauluista voidaan lukea maksimissaan 1000 riviä yhdellä transaktiolla, joten 10 000 viittauksen hakemiseen kuluu optimaalisessa tilanteessa kymmenen transaktiota. StorageClient-kirjasto osaa automaattisesti jakaa kyselyn osiin ja yhdistää tulosjou-

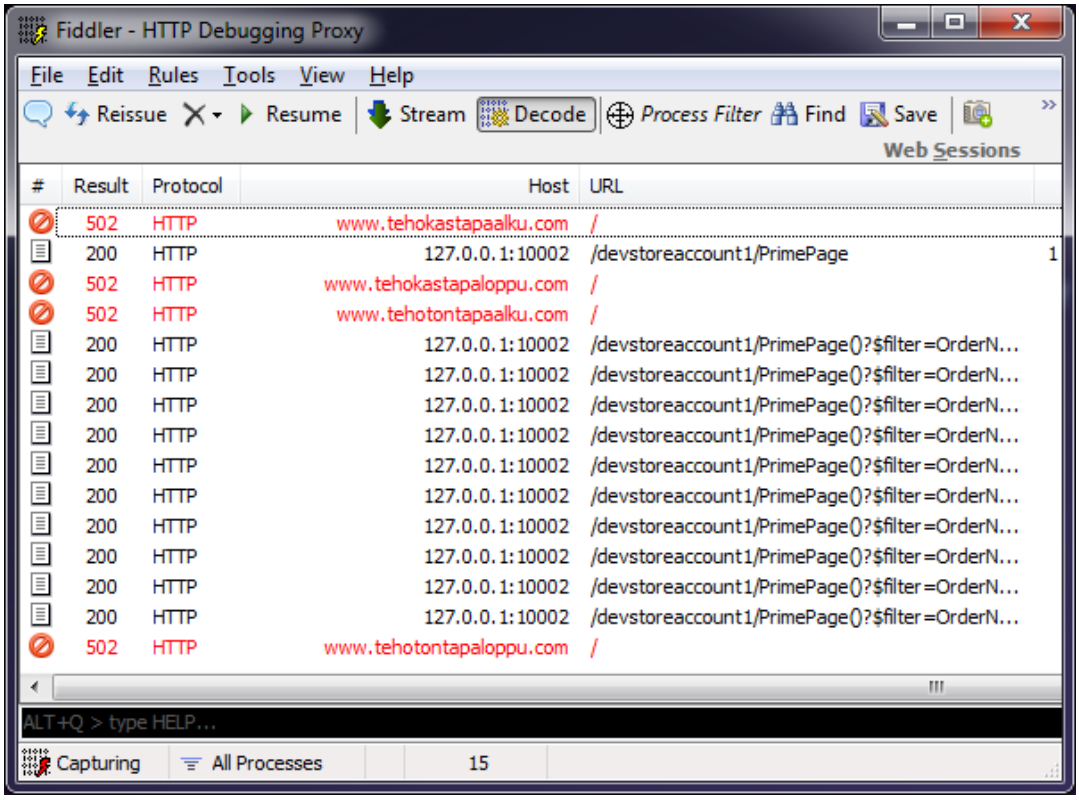
kon. Kannattavinta on ladata taulun tiedot StorageClient-kirjaston avulla suoraan .Net:n List-kokoelmaan. Näin varmistetaan, että kaikki tiedot ladataan kerralla. Ehdottomasti tulisi välttää tilannetta, jossa StorageClient-kirjaston tarjoamaan kokoelmaa taulun riveistä käytetään useita kertoja esimerkiksi silmukan sisällä. Tällöin saattaa käydä niin, että jokaista silmukan kierrosta kohti muodostuu transaktio. Esimerkkikoodi 1 sisältää simulaatio-ohjelman, jossa esitellään kaksi tapaa lukea taulun rivejä.

```
[TestMethod()]
   public void HaeRivejaSimulaatio()
   {
   //1. Tehokas tapa
       //Haetaan kaikki taulurivit ensin muistiin .ToList() metodilla
       Tr("http://www.tehokasTapaAlku.com");
       var tauluRivitLista = new PageEntityContext(account).Pages.ToList();
       //Simuloidaan kahden taulun vertaamista toisiinsa
       for (int i = 1; i <= 10; i++)
       {
           //Poimitaan ensimmäinen rivi, jonka järjestysnumero-
           //ominaisuus(OrderNumber) vastaa juoksevaa lukua i
           var ensimmainenRivi = tauluRivitLista
                                       .Where(rivi => rivi.OrderNumber == i)
                                       .FirstOrDefault();
       }
       Tr("http://www.tehokasTapaLoppu.com");

   //2. Tehoton tapa
       //Käytetään taulun rivejä suoraan StorageClientin kautta, ei ladata kaikkia
       //rivejä ensin muistiin
       Tr("http://www.tehotonTapaAlku.com");
       var tauluRivitStorageClient = new PageEntityContext(account).Pages;
       for (int i = 1; i <= 10; i++)
       {
           var ensimmainenRivi = tauluRivitStorageClient
                                       .Where(rivi => rivi.OrderNumber == i)
                                       .FirstOrDefault();
       }
       Tr("http://www.tehotonTapaLoppu.com");
   }
```

Esimerkkikoodi 1. Taulun rivien lukeminen kahdella eri tavalla.

Esimerkissä simuloidaan tilannetta, jossa taulusta haetaan tietoa silmukan sisällä. Tapauksessa 1 kaikki taulun rivit luetaan kerralla muistiin kutsumalla StorageClientin mallintamalle rivikokoelmalle (Pages) ToList-metodia. Tällaisessa tapauksessa kaikki rivit ladataan tuhannen erissä kerralla muistiin. Tapaus 2 on muuten täysin identtinen, mutta siinä taulun rivejä ei ladata kerralla muistiin. Molemmissa tavoissa käydään kymmenen kertaa silmukan sisällä etsimässä tietoa taulun riveistä. Koska tapauksessa 2 ei ladata taulun rivejä suoraan muistiin, muodostuu jokaisella silmukan kierroksella tallujärjestelmään transaktio. Kuviossa 22 esitetään kaappaus http-pyynnöistä, kun simulaatio-ohjelma suoritetaan.



Kuvio 22. Kaappaus http-pyynnöistä jotka syntyvät kun simulaatio-ohjelma ajetaan.

Kuvassa punaiset rivit kuvaavat simulaatio-ohjelman eri toteutustapojen alkua ja loppua. Mustat rivit osoittavat Windows Azure -tallennusjärjestelmiin tehtyjä http-pyyntöjä. Jokainen näistä pyynnöistä muodostaa yhden transaktion Azuren tallennusjärjestelmään. Kaappauksesta on hyvin havaittavissa se, että ensimmäisellä toteutustavalla tallennusjärjestelmään muodostuu ainoastaan yksi transaktio. Toisessa toteutustavassa saman asian tekemiseen kuluu kymmenen transaktiota. Esimerkistä huomaa selkeästi sen, että melko pienellä toteutustavan muutoksella saattaa olla dramaattinen vaikutus käytettyyn transaktiomäärään ja sitä myöten palvelun käytöstä muodostuviin kustannuksiin. Esimerkissä taulun riveistä haettiin tietoa ainoastaan kymmenen kertaa. Näin ollen, jos tässä tilanteessa tietoa haettaisiin miljoona kertaa, muodostuisi tehokkaan tavan transaktiokustannuksiksi 0,00001 $. Vastaavasti tehottoman tavan transaktiokustannukset olisivat 1 $. Edellä käytetty menetelmä tutkia transaktioiden määrää http-kaappauksen avulla osoittautui yksinkertaiseksi ja tehokkaaksi tavaksi havaita mahdollisia ongelmakohtia sovelluksen toiminnassa.

Tietoja julkaistaessa syntyy taulujärjestelmään suuri määrä tallennuksia. Tämän tyyppisissä tilanteissa kannattaa käyttää hyväksi mahdollisuutta tallentaa tietoja isoissa erissä. Azuren taulujärjestelmä sallii 100 rivin tallentamisen kerrallaan, jolloin siihen käytetään ainoastaan yksi transaktio. Tämän ominaisuuden hyödyntäminen Storage-Client-kirjaston avulla vaatii sitä, että taulua ei tallenneta jokaisen lisäyksen jälkeen, vaan aina, kun siihen on lisätty 100 riviä.

5.3 Testaus

Tiedonsiirtomoduulille määriteltiin joukko testitapauksia, jotka sen tulee tämän työn puitteissa läpäistä. Testitapauksilla pyritään varmistamaan, että järjestelmä toimii määrittelyjen mukaisesti. Jokaista testitapausta kohti ohjelmoitiin testi, joka varmistaa ohjelmiston toiminnan.

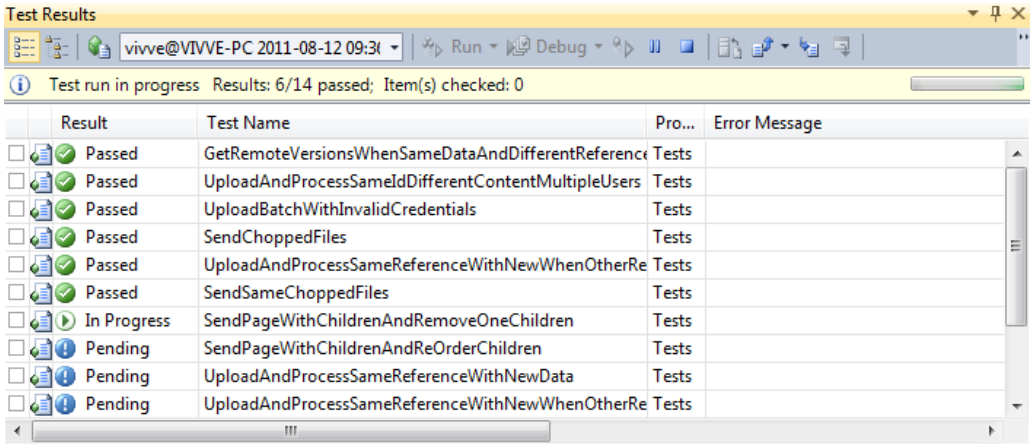
Esimerkkejä määritellyistä testitapauksista ovat seuraavat:

- synkronointitietojen pyytäminen
- kokonaisten tiedostojen lähettäminen ja julkaisu
- pilkottujen tiedostojen lähettäminen ja julkaisu
- saman tiedon lähettäminen ja julkaisu useaan kertaan eri nimillä
- rajapinnan käyttäminen väärillä käyttäjätunnuksilla.

Ohjelmistokehityksessä käytetään useita eri testityyppejä. Yleisiä testityyppejä ovat muun muassa yksikkötestit, integraatiotestit, rasitustestit ja hyväksymistestit. Yksikkötesteillä tutkitaan ohjelmiston toimintaa mahdollisimman pienissä osissa. Tyypillisesti yksikkötestissä tutkitaan esimerkiksi, miten luokan tietty metodi toimii eri syötteillä. Integraatiotestillä tarkoitetaan tilannetta, jossa testataan usean ohjelmisto-osan tai järjestelmän toimintaa ja varmistetaan, että osat toimivat yhdessä. Rasitustestien tarkoituksena on tutkia järjestelmän käyttäytymistä kun siihen kohdistetaan vaihtelevia kuormia. Hyväksymistestaus on yleensä asiakkaan suorittama testaus jossa tarkastetaan, että järjestelmä toimii odotetulla tavalla oikeissa käyttötilanteissa [31, s. 20–24]. Tiedonsiirtomoduulin testit testaavat useiden toiminnallisuuksien yhdistelmiä, esimerkiksi tiedostojen lähetys ja julkaisu. Lisäksi osa testattavista ominaisuuksista liittyy koko

pilvijulkaisualustaan toteutukseen. Esimerkiksi tietojen lopullisen tallennuspaikan julkaisun jälkeen määrittelee pilvijulkaisualusta eikä pelkkä tiedonsiirtomoduuli. Tästä syystä toteutettuja testejä voidaan pitää integraatiotesteinä.

Testaus toteutettiin Visual Studio 2010 -ohjelmistokehitysympäristön avulla. Visual Studio sisältää mahdollisuuden luoda testausprojekteja, joiden avulla testaus suoritetaan. Testausprojektiin ohjelmoidaan halutut testit, jonka jälkeen projektiin luotuja testejä voidaan ajaa aina tarvittaessa. Tiedonsiirtomoduulin testauksessa testit matkivat lähdejärjestelmää kutsuen ohjelmiston julkisen rajapinnan operaatioita. Testaaminen osoittautui erittäin hyödylliseksi toimenpiteeksi sovellusta kehitettäessä. Testejä voitiin ajaa toistuvasti sovellusta kehitettäessä. Näin havaittiin nopeasti mahdolliset ongelmat joita sovellukseen tehdyt muutokset aiheuttivat. Kuviossa 23 suoritetaan tiedonsiirtomoduulille testejä Visual Studion avulla.



Kuvio 23. Visual Studion testinäkymä.

Kuviossa näkyy riveittäin listattuna suoritettavat testit. Testeistä kuusi ensimmäistä on suoritettu, ja ne ovat tuottaneet hyväksytyn tuloksen. Seitsemännen testin suoritus on kesken ja loput odottavat suorittamista. Testit testaavat muun muassa pilkottujen tiedostojen lähettämistä ja väärien käyttäjätunnusten käyttöä.

6 Yhteenveto

Erilaiset pilvijärjestelmät kasvattavat suosiotaan jatkuvasti. Syitä niiden suosioon ovat muun muassa käyttöperusteinen hinnoittelu ja hyvä skaalautuvuus. Erityisesti käyttöperusteinen hinnoittelu tekee niistä erittäin houkuttelevan vaihtoehdon startup-yrityksille. Käyttämällä pilvijärjestelmiä vältytään suurilta infrastruktuuri-investoinneilta ja resursseja voidaan käyttää enemmän mahdollisen hyvän idean kehittämiseen. Pilvijärjestelmiä on tarjolla useilta eri yrityksiltä, ja niiden merkittävimmät erot liittyvät lähinnä siihen, kuinka paljon järjestelmän asetuksiin voi itse vaikuttaa. Lisäksi eri pilvijärjestelmät tukevat eri ohjelmointikieliä ja teknologioita. Nämä seikat vaikuttavat luonnollisesti käytettävän järjestelmän valintaan.

Insinöörityössä suunniteltiin ja toteutettiin tiedonsiirtomoduuli Microsoftin Windows Azure Platform -pilvipalvelualustalle. Tiedonsiirtomoduulin toteuttamiseksi harkittiin muutamia valmiita ratkaisuja, mutta näiden puutteista johtuen ohjelmisto päätettiin toteuttaa kokonaan itse. Ratkaisussa ohjelmiston internetissä näkyvä julkinen rajapinta toteutettiin WWW-sovelluspalveluna. Rajapinnan toteutuksessa käytettiin Windows Communication Foundation (WCF) -alustaa, joka teki toteutuksen melko helpoksi. Suurin haaste WCF:n käytössä oli sen konfiguroiminen siten, että tietoturva on tarpeeksi hyvä ja että sen välityksellä pystytään siirtämään riittävän isoja tiedostoja. WWW-sovelluspalvelun toteuttaminen Windows Azureen oli periaatteessa mutkatonta, mutta Azuren hieman keskeneräinen dokumentaatio hidasti työtä paikoitellen.

Ilmeinen muutos sovellusten kehittämisessä pilvijärjestelmille on järjestelmän käytöstä aiheutuvat kustannukset. Näiden kustannusten muodostuminen tulee huomioida koko sovelluskehityksen aikana suunnittelusta alkaen. Tutkimuksessa saatiin esille haasteita, joita sovellusten kehittäminen Windows Azure Platform alustalle tuo. Nämä haasteet pätevät luultavasti myös moniin muihin pilvijärjestelmiin. Suurin muutos perinteiseen sovelluskehitykseen on se, että sovelluskehittäjän tekemillä ratkaisuilla saattaa olla merkittävä vaikutus sovelluksen käytöstä laskutettavaan hintaan. Koska alustan käyttö on hinnoiteltu käyttömääräperusteisesti, vaikuttavat yksittäiset tekniset ratkaisut siihen, kuinka kalliiksi sovelluksen käyttö muodostuu. Yksi merkittävimmistä kohteista, joka aiheuttaa vaihtelevia kuluja toteutustavasta riippuen, on tallennusratkaisuiden käyttö-

tapa. Tämän lisäksi tallennusratkaisuiden valinnalla voidaan vaikuttaa kulujen muodostumiseen. Tämä koskee esimerkiksi valintaa Azure-taulujen ja SQL Azuren välillä.

Käteväksi menetelmäksi tallennusratkaisujen käytön arvioinnissa osoittautui tallennusjärjestelmään lähetettyjen http-pyyntöjen kaappaaminen ja analysointi. Analysoinnin ja koodin tulkitsemisen avulla voidaan havaita mahdollisia tallennusjärjestelmään tulevia turhia pyyntöjä, jotka kasvattavat sovelluksen käytöstä aiheutuvia kuluja. Käyttömääräperusteinen hinnoittelu johtaa siihen, että sovelluksia pyritään optimoimaan mahdollisimman edulliseksi käyttää. Sovellusten optimointia ei kuitenkaan tule tehdä rakenteen ja sovelluksen selkeyden kustannuksella vaan sovelluksen ylläpidettävyys tulee myös pitää mielessä. Sovelluskehitys Microsoftin pilvipalvelualustalle ei vaatinut juurikaan uusien teknologioiden opiskelua, joten sen omaksuminen sovelluskehittäjän näkökulmasta on melko helppoa ja nopeaa.

Pilvijulkaisualusta, jonka osana insinöörityössä toteutettu tiedonsiirtomoduuli toimii, esiteltiin Los Angelesissa Microsoftin kumppanuustapahtumassa heinäkuussa 2011. Järjestelmä sai positiivisen vastaanoton. Julkaisualustaan on myös yhdistetty ensimmäinen tiedonsiirtomoduulia hyödyntävä julkaisujärjestelmä. Tämän integraation avulla julkaisujärjestelmästä voidaan julkaista sisältöä pilvijulkaisualustalle. Pilvijulkaisualustan ja tiedonsiirtomoduulin kehitystä jatketaan vielä.

Lähteet

1 Caceres, J., Linder, M., Rodero-Merino, L., Vaquero L. M. A Break in the Clouds: Towards a Cloud Definition. 2009. Verkkodokumentti. ACM SIGCOMM Computer Communication Review. [http://ccr.sigcomm.org/online/files/p50-v39n1l-vaqueroA.pdf] Luettu 3.7.2011.

2 Krishnan, Sriram. 2010. Programming Windows Azure. Sebastopol: O`Reilly Media.

3 Brewer, Eric A. Towards robust distributed systems. 2000. Verkkodokumentti. UC Berkley. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf Luettu 10.9.2011.

4 Broberg, J., Buyya, R., Goscinski, A. 2011. Cloud compiting principles and paradigms. New Jersey: John Wiley & Sons.

5 Ahson, Syed A., Ilyas Mohammad. 2011. Cloud computing and software services. Boca Baton: CRC Press.

6 Kirby J., Stewart, T. A. The Institutional Yes. Verkkodokumentti. Harward Business review. [http://hbr.org/2007/10/the-institutional-yes/ar/pr] Luettu 3.7.2011.

7 Paganelli, F. & Vliet J. 2011. Programming Amazon EC2. Sebastopol: O`Reilly Media.

8 Douglas, J. & Roche K. 2009. Beginning Java Google App Engine. New York: Springer-Verlag.

9 Revision history. 2011. Verkkodokumentti. Google. [http://code.google.com/appengine/docs/revision_history.html] Luettu 10.7.2011.

10 What is Google App Engine? Verkkodokumentti. Google. [http://code.google.com/appengine/docs/whatisgoogleappengine.html] Luettu 9.7.2011.

11 Billing and budgeting resources. Verkkodokumentti. Google. [http://code.google.com/appengine/docs/billing.html] Luettu 9.7.2011.

12 Windows Azure Team. Windows Azure Platform launch update. 2009. Verkkodokumentti. Microsoft. [http://blogs.msdn.com/b/windowsazure/archive/2009/10/29/windows-azure-platform-launch-update.aspx] Luettu 31.7.2011.

13 Brunetti, Roberto. 2011. Windows Azue Step by Step. Sebastopol: O`Reilly Media.

14 Hay, C., Prince, B. H. 2011. Azure in action. Stamford: Manning Publications Co.

15 Li, Henry. 2009. Introducing Windows Azure. New York: Springer-Verlag.

16 Service Bus. Verkkodokumentti. Microsoft. [http://www.microsoft.com/windowsazure/features/servicebus] Luettu 10.9.2011.

17 Pricing Overview. Verkkodokumentti. Microsoft. [http://www.microsoft.com/windowsazure/pricing] Luettu 10.9.2011.

18 Lampi, Mikko. 2010. Varapääjohtaja, Innofactor Oyj, Espoo. Keskustelu 22.12.2010.

19 Ärje, Matias. 2010. Johtaja, Innofactor Oyj, Espoo. Keskustelu 22.12.2010.

20 Ärje, Matias. 2011. Johtaja, Innofactor Oyj, Espoo. Keskustelu 22.1.2011.

21 Ärje, Matias. 2011. Johtaja, Innofactor Oyj, Espoo. Keskustelu 29.3.2011.

22 Introduction to Microsoft Sync Framework. 2009. Verkkodokumentti. Microsoft. [http://msdn.microsoft.com/en-us/sync/bb821992] Luettu 1.8.2011.

23 SQL Azure Data Sync - How to get started. 2011. Verkkodokumentti. Microsoft. [http://social.technet.microsoft.com/wiki/contents/articles/2197.aspx] Luettu 28.8.2011.

24 Mulder, D., Peiris, C. 2007. Pro WCF practical Microsoft SOA implementation. New York: Springer-Verlag.

25 Northrup, Tony. 2009. Microsoft .NET framework–application developement foundation. Washington: Microsoft Press.

26 Talbot, J. & Welsh, D. 2006. Complexity and cryptography an introduction. New York: Cambridge University Press.

27 Johnson, B., Madziak, P., Morgan, S. 2009. Microsoft .NET framework 3.5–Windows Communication Foundation. Washington: Microsoft Press.

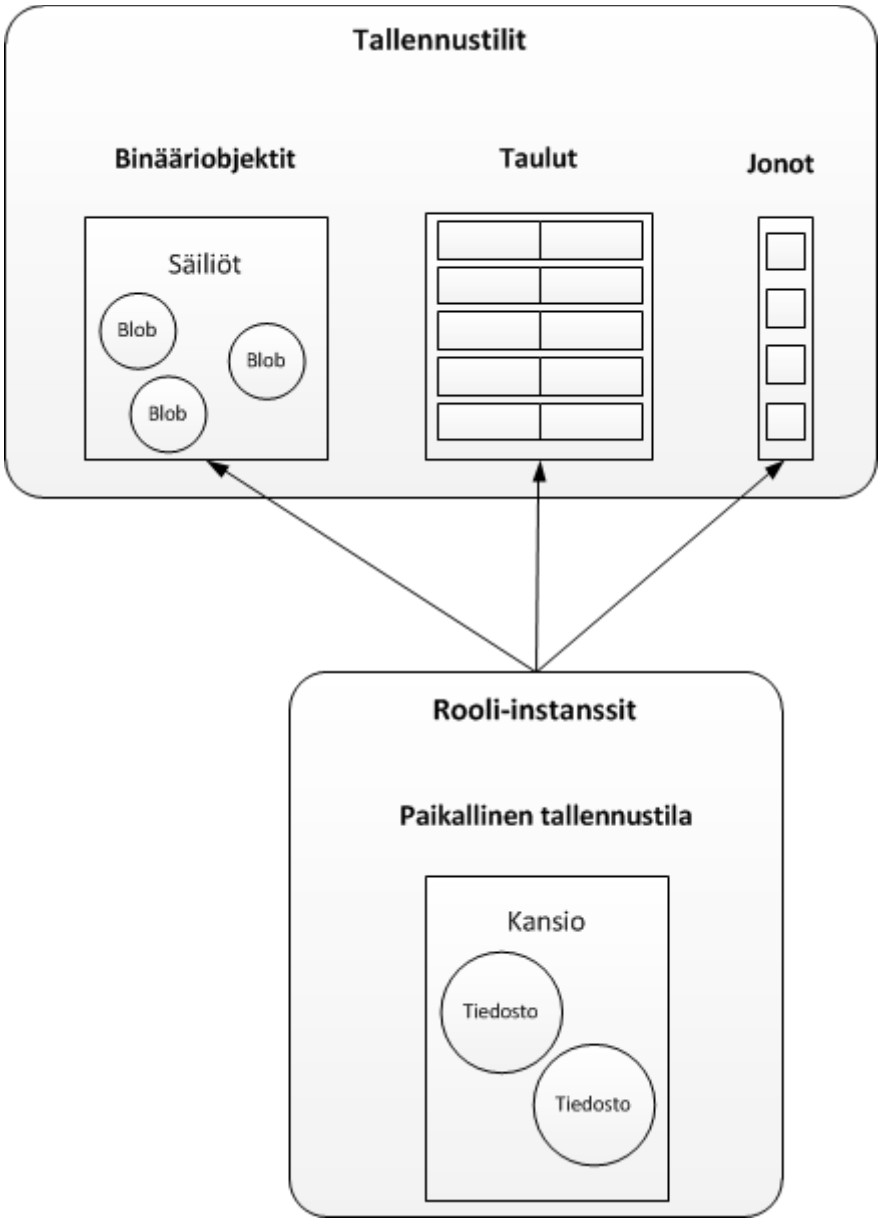
28 Cibraro, P., Claeys, K., Cozzolino, F., Grabner, J. 2010. Professional WCF 4. Indiana: Wiley Publishing.

29 Wang, Brandon J. WCF Azure Worker Role on HTTP port 80. 2011. Verkkodokumentti. Microsoft. [http://code.msdn.microsoft.com/WCF-Azure-Worker-Role-on-b394df49] Luettu 29.8.2011.

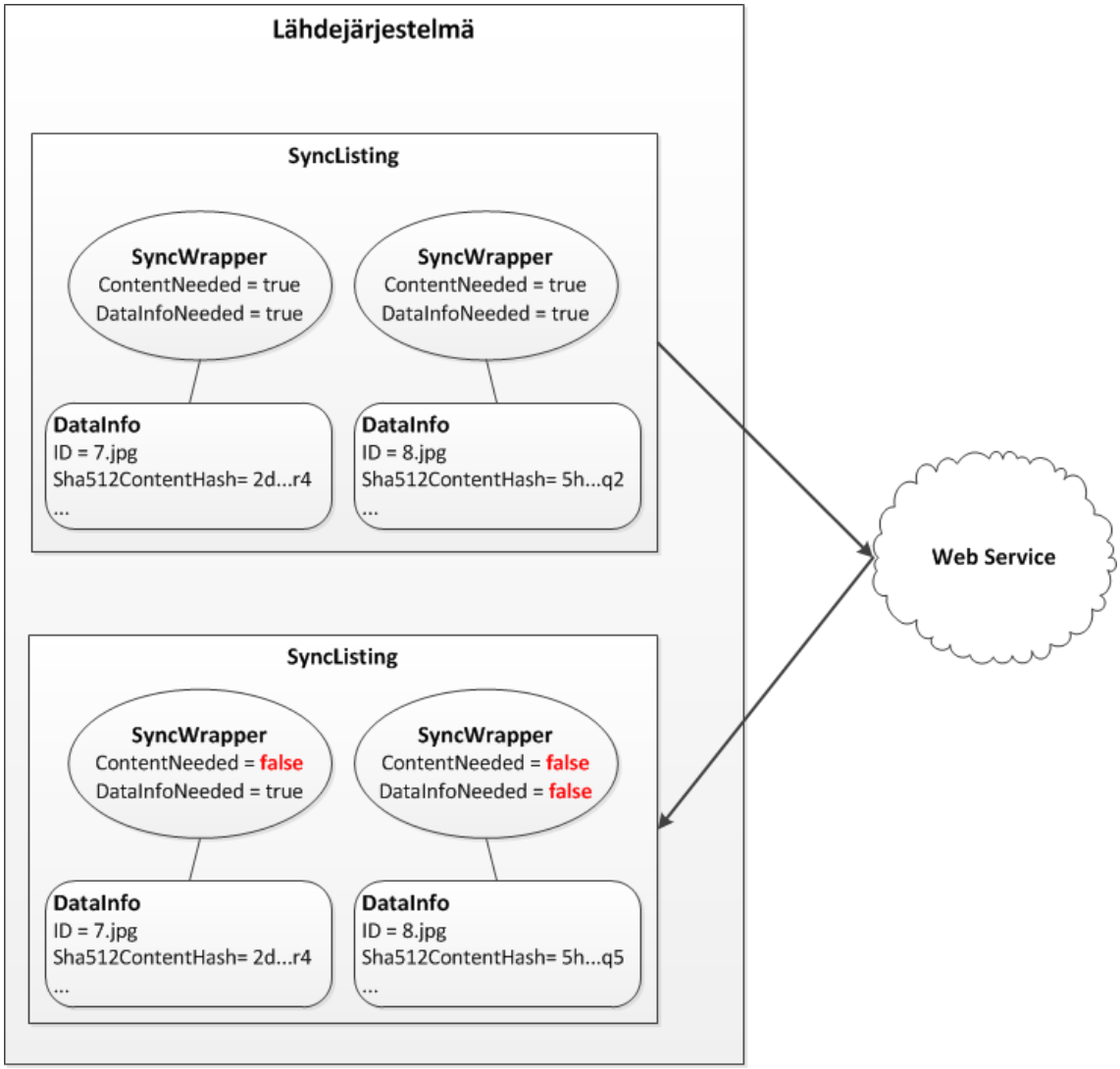
30 Bishop, Judith. 2008. C# 3.0 design patterns. Sebastopol: O`Reilly Media.

31 Bender, J., McWherter, J. 2011. Professional test driven development with C#. Indiana: Wiley Publishing.

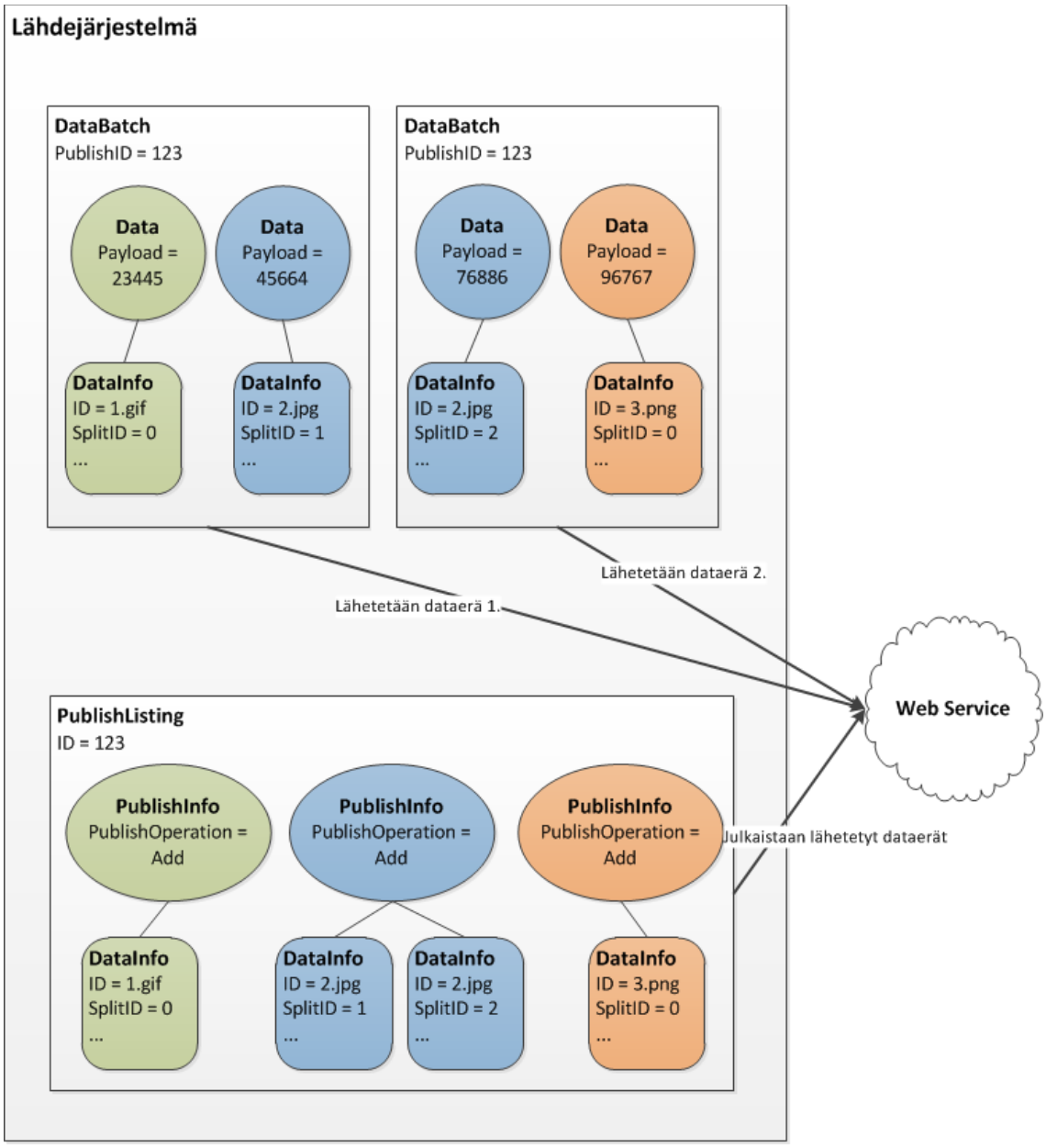
Windows Azure -tallennusjärjestelmät



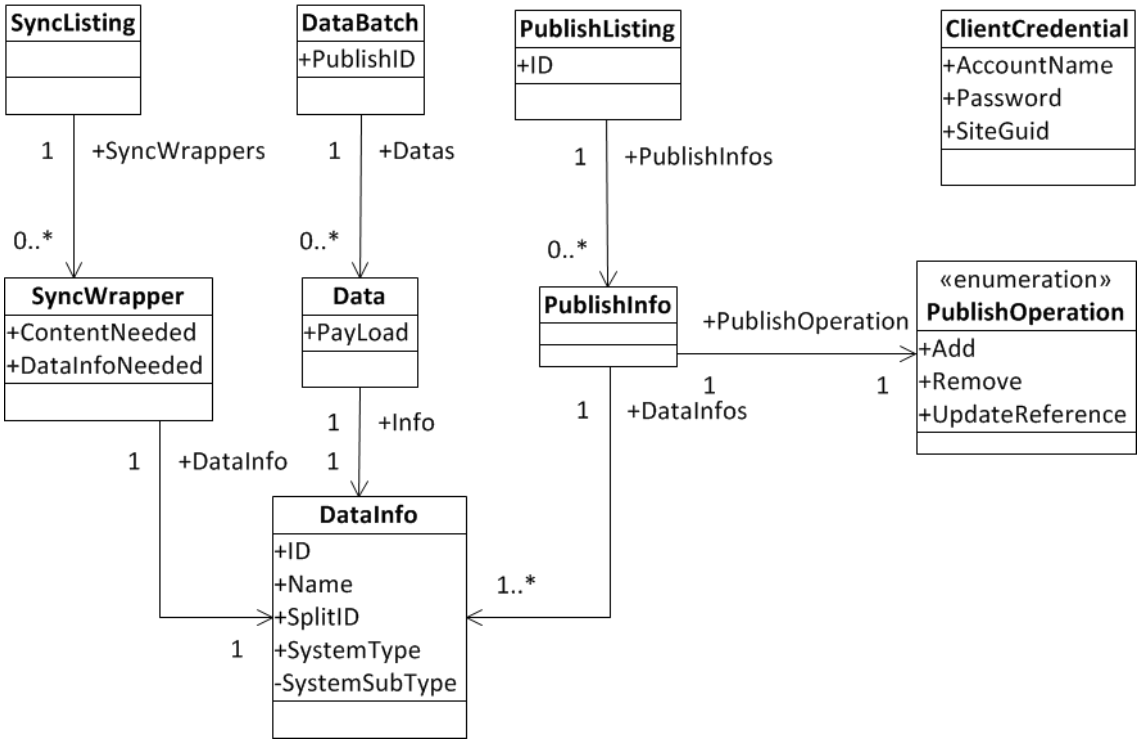
Synkronointitietojen pyytäminen



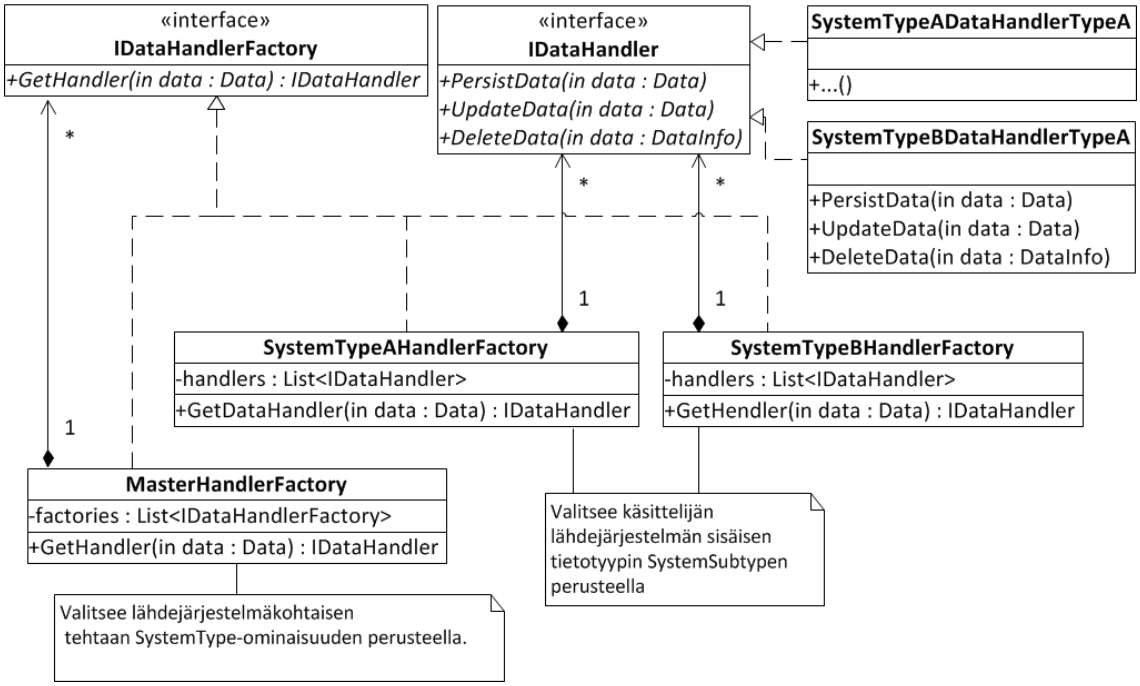
Tietojen siirto erissä ja niiden julkaisu



Tiedonsiirtomoduulin viestit ja niiden suhteet



Tehdas-suunnittelumallin hyödyntäminen



Käsittelijän hakeminen lähdejärjestelmän A tietotyypille B